ETH zürich

12.11.2020

PVK HS2020

Josephine Loehle
jloehle@student.ethz.ch

Organisatorisches

- Über mich: Josephine, 3. Semester Elektrotechnik
- Webseite: https://www.n.ethz.ch/student/jloehle/
  - Folien, Zusammenfassung, Notizen
- Email: jloehle@student.ethz.ch

Ablauf

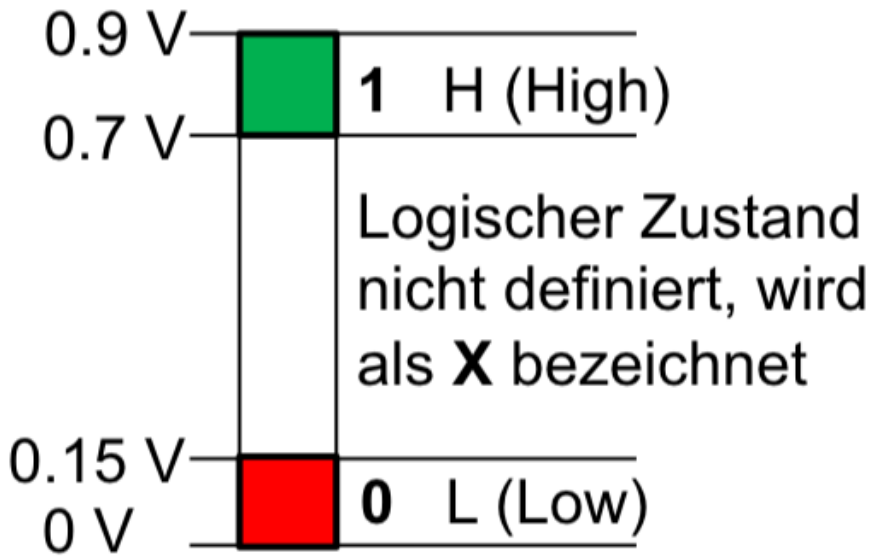
- 9:00-10:30: Theorie 1
- Kurze Pause
- 10:40-12:00: Theorie 2
- Mittagspause
- 13:00-16:00: Prüfung lösen + Zeit für Fragen

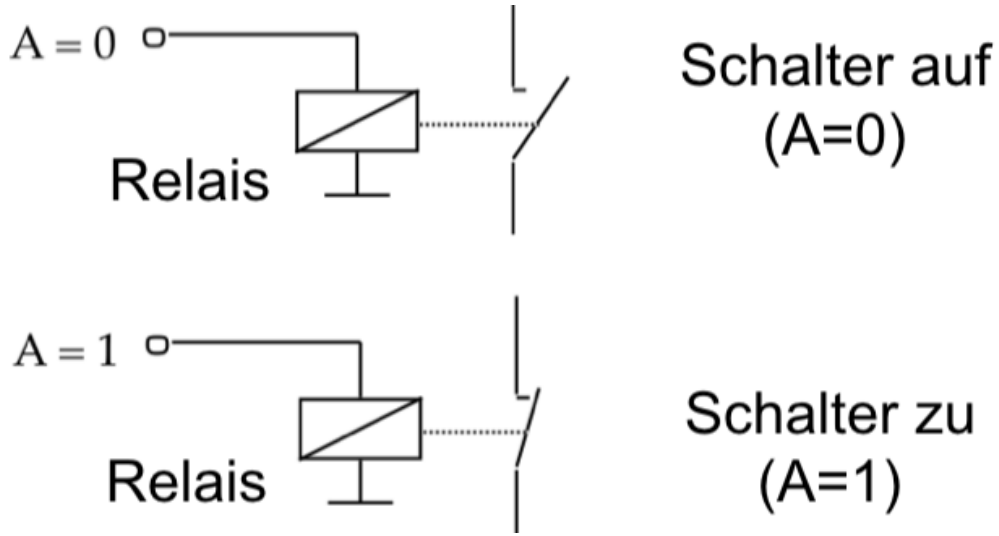
Logische Verknüpfungen

Zweiwertige Logik – Bits

Bits

- 2 binäre Zustände: 0 & 1
- n Bits können 2^n Zustände beschreiben
- MSB … LSB





Wahrheitstabellen

- N Eingänge
- M Ausgänge

2^N Zeilen

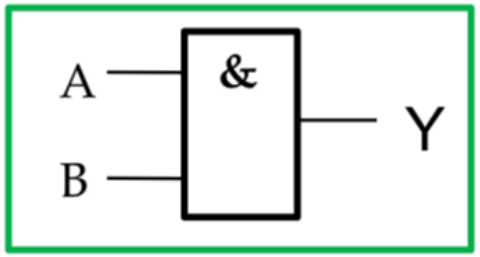
A	B	C	Y
0	0	0	$f(0,0,0) \in \{0,1\}$
0	0	1	$f(0,0,1) \in \{0,1\}$
0	1	0	$f(0,1,0) \in \{0,1\}$
0	1	1	$f(0,1,1) \in \{0,1\}$
1	0	0	$f(1,0,0) \in \{0,1\}$
1	0	1	$f(1,0,1) \in \{0,1\}$
1	1	0	$f(1,1,0) \in \{0,1\}$
1	1	1	$f(1,1,1) \in \{0,1\}$

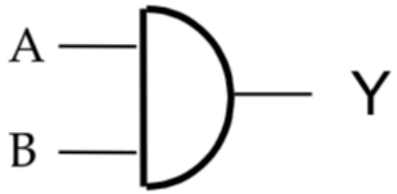
M + N Spalten

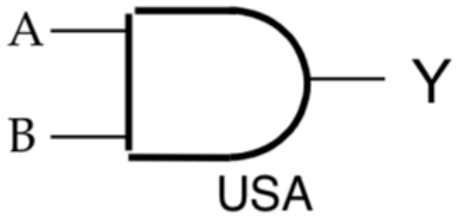
UND-Verknüpfung

- Y = 1, wenn A = 1 & B = 1

Hier verwendet







A	B	Y
0	0	
0	1	
1	0	
1	1	

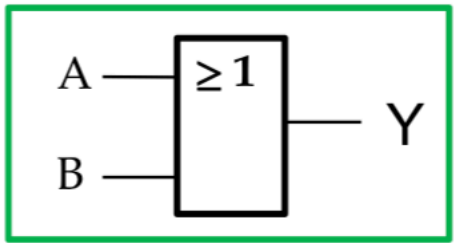
$$Y = A \wedge B \qquad Y = A \cdot B$$

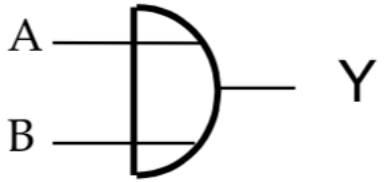
Hier verwendet

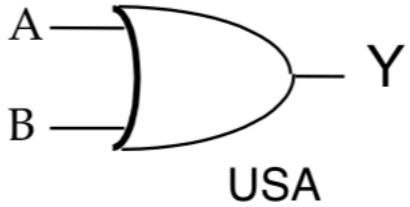
ODER-Verknüpfung

- Y = 1, wenn A = 1 &/oder B = 1

Hier verwendet





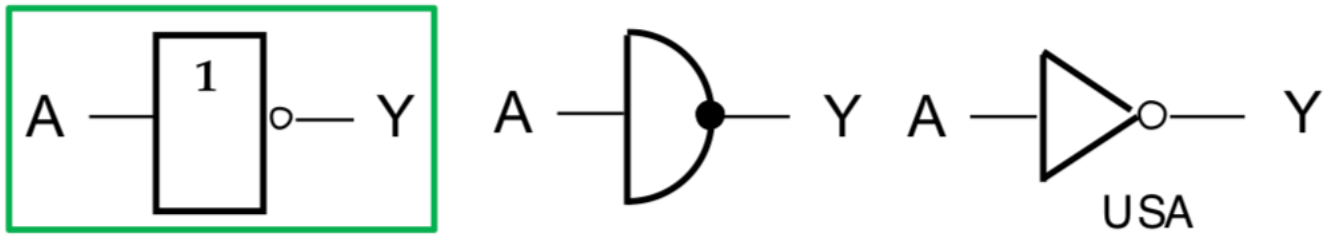


A	B	Y
0	0	
0	1	
1	0	
1	1	

$$Y = A \vee B \qquad Y = A + B$$

INVERTER

- Y = 1, wenn A = 0

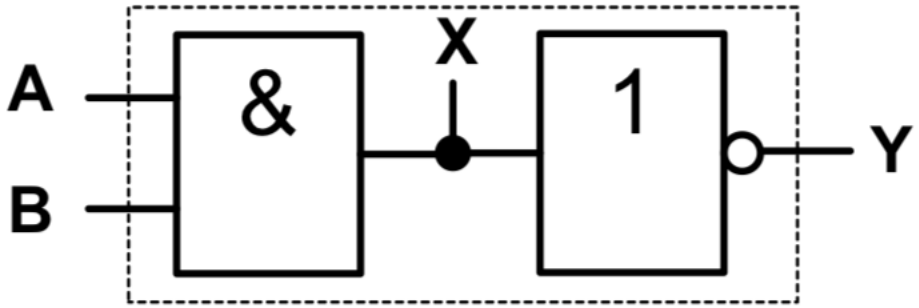


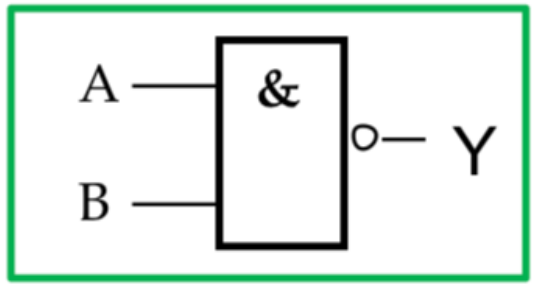
A	Y
0	
1	

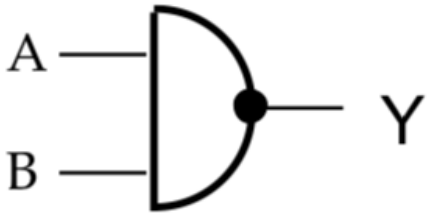
Hier verwendet $Y = \overline{A}$

NAND-Gatter

- Y = 1, wenn A = 0 &/oder B = 0







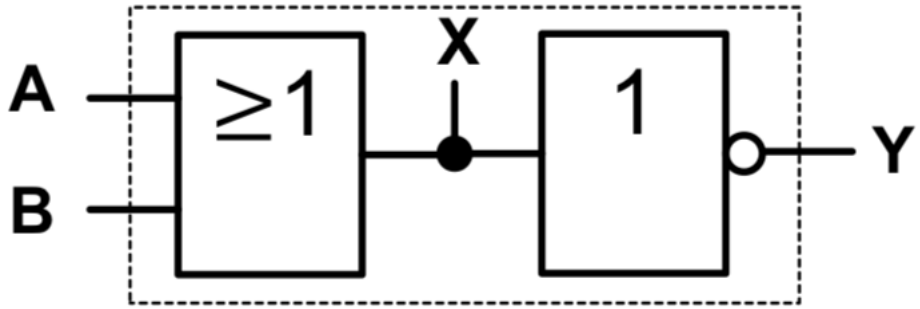
A	B	X	Y
0	0		
0	1		
1	0		
1	1		

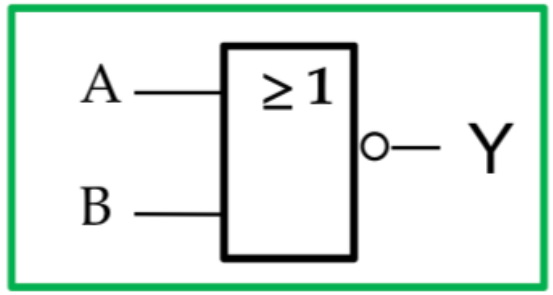
$$Y = \overline{A \wedge B} \qquad Y = \overline{A \cdot B}$$

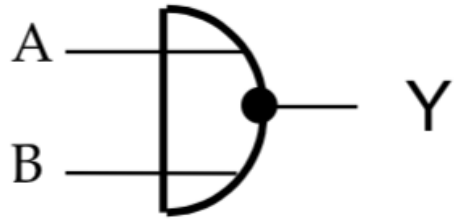
Hier verwendet

NOR-Gatter

- Y = 1, wenn A = 0 & B = 0







A	B	X	Y
0	0		
0	1		
1	0		
1	1		

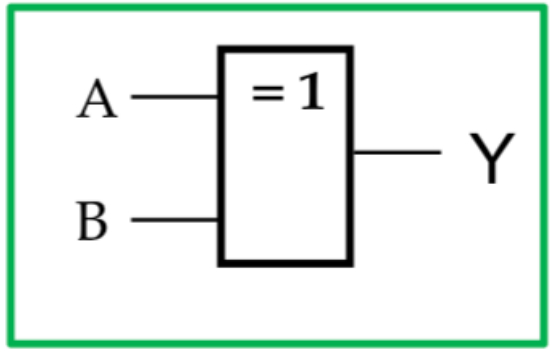
$$Y = \overline{A \vee B} \qquad Y = \overline{A + B}$$

XOR-Gatter (= Exclusive Or)

- Y = 1, wenn A = 0 oder B = 0

A	B	X
0	0	
0	1	
1	0	
1	1	

Hier verwendet



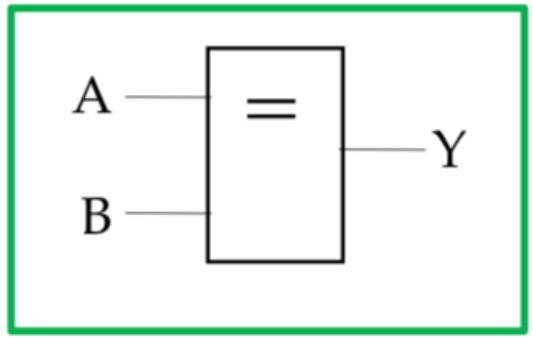
Hier verwendet

$$Y = A \oplus B$$

XNOR-Gatter

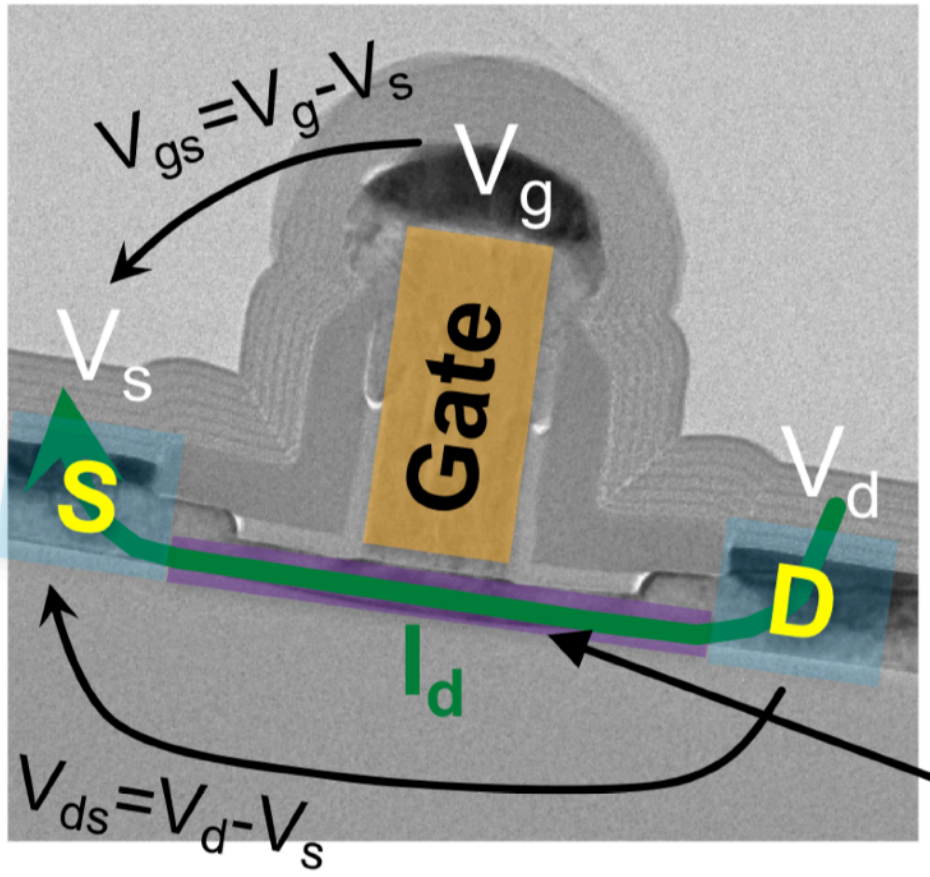
- Invertiertes XOR-Gatter

$$Y = \overline{A \oplus B}$$



CMOS Schaltungen

MOS-Transistoren



Source

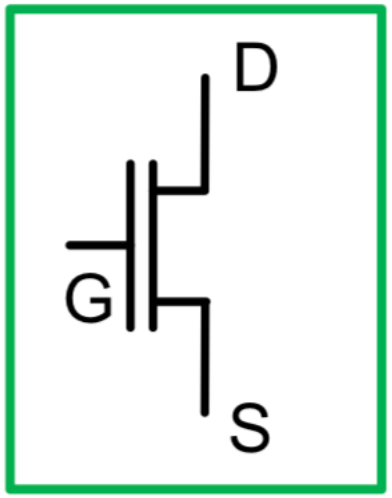
Drain

R_T (Halbleiter)

- |V_gs| < |V_th| kein Strom fliesst
- |V_gs| > |V_th| es fliesst Strom
- |V_th| = Schwellenspannung (ca. 0.8V)
- Funktioniert durch Halbleiter

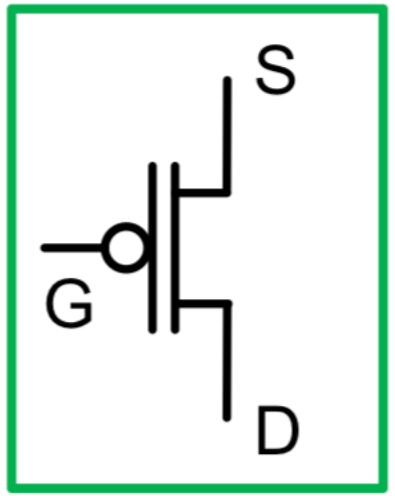
NMOS

- Elektronen fliessen von S zu D
- P-Typ Halbleiter
- Für Pull-down Schaltung verwendet



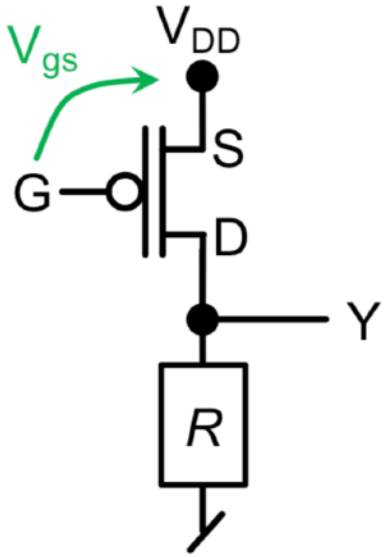
PMOS

- Löcher fliessen von S zu D
- N-Typ Halbleiter
- Für Pull-up Schaltung verwendet



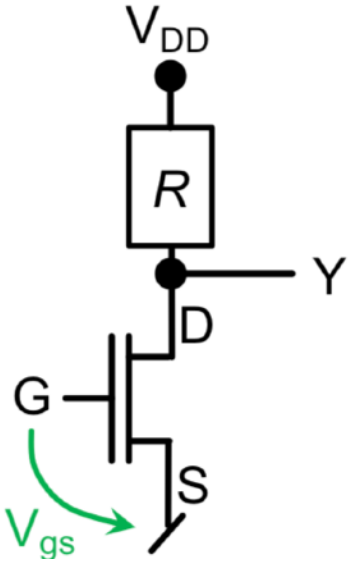
Pull-up Schaltung

- PMOS-Transistoren
- Leitet, wenn G != S (G = 0)
- Leitet eine 1 weiter (an V_DD gebunden)
- NN (Unbestimmter Zustand) wenn Transistor davor gesperrt



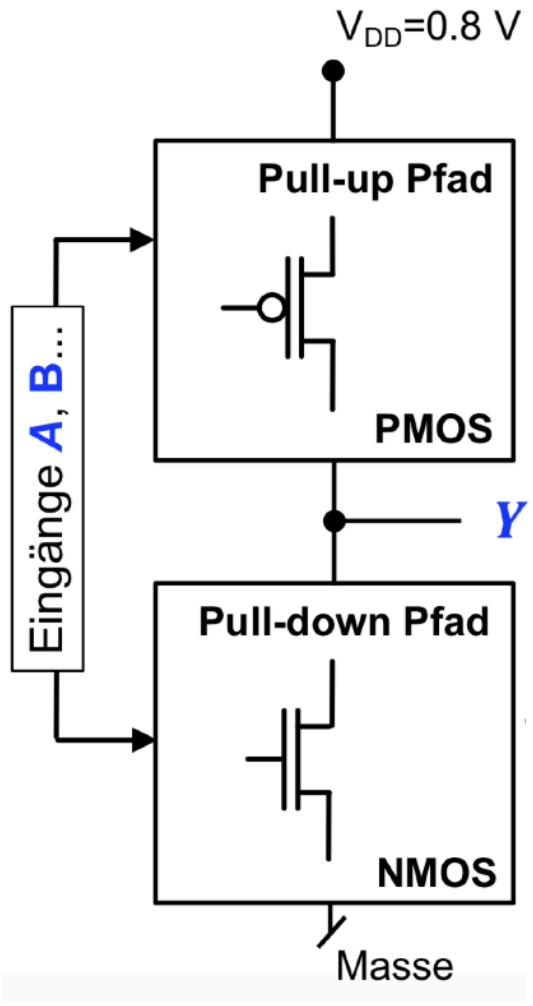
Pull-down Schaltung

- NMOS-Transistoren
- Leitet, wenn G != S (G = 1)
- Leitet eine 0 weiter (an Masse gebunden)
- NN (Unbestimmter Zustand) wenn Transistor davor gesperrt

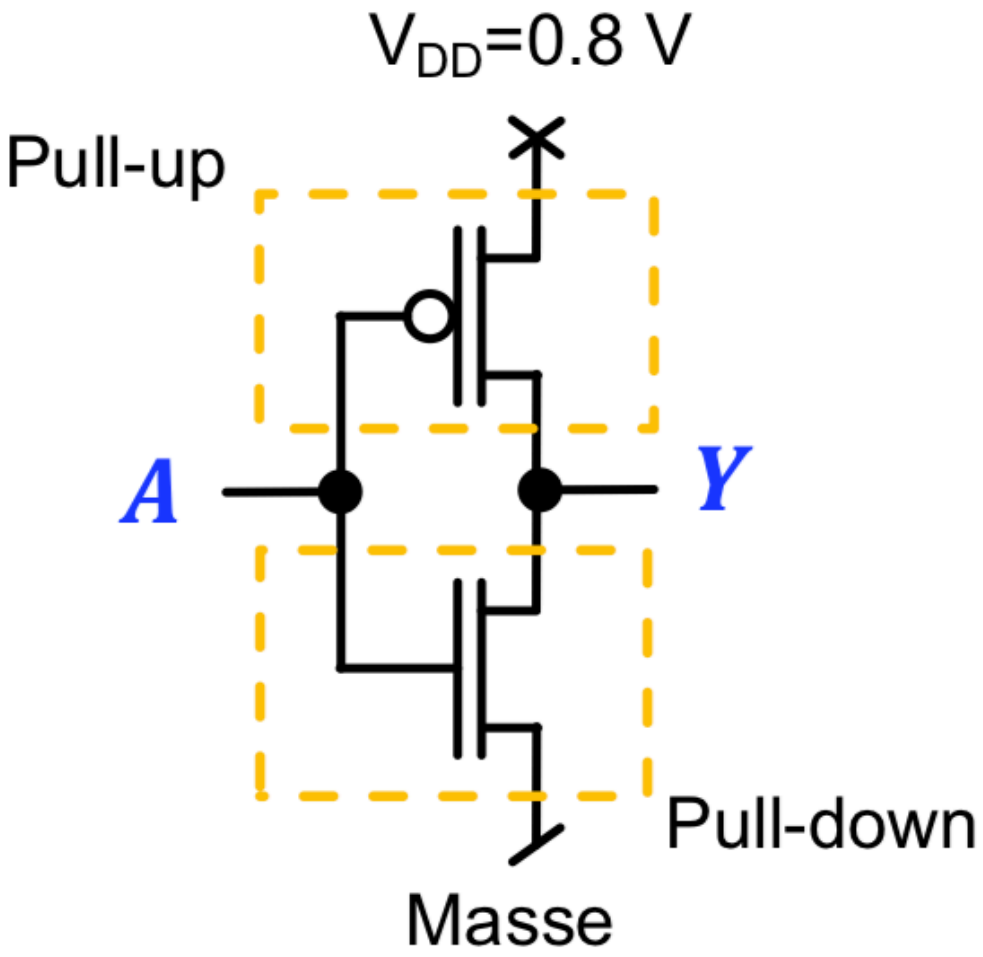


Schaltungen

- # NMOS-Transistoren = # PMOS-Transistoren = # Eingänge

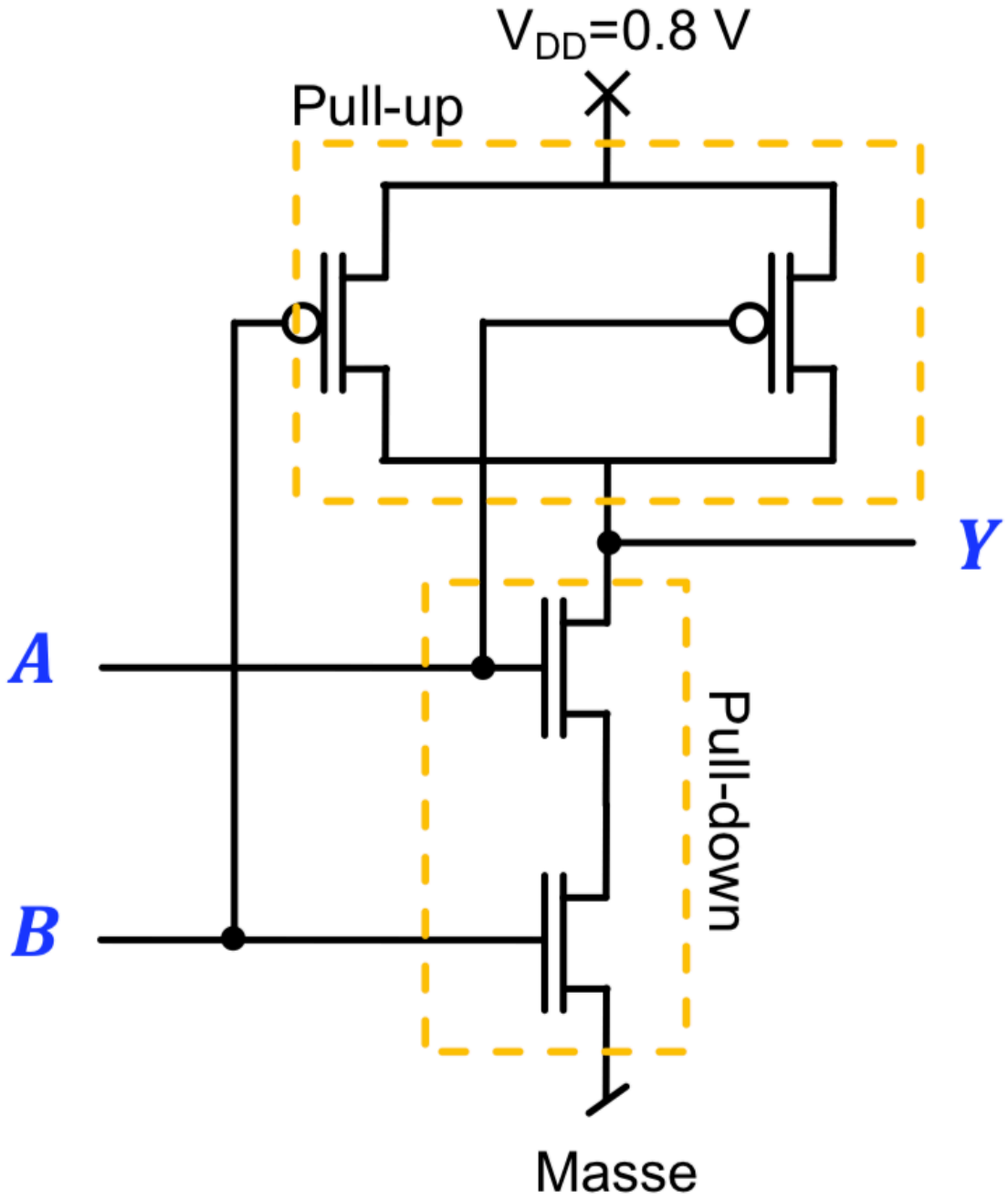


NOT



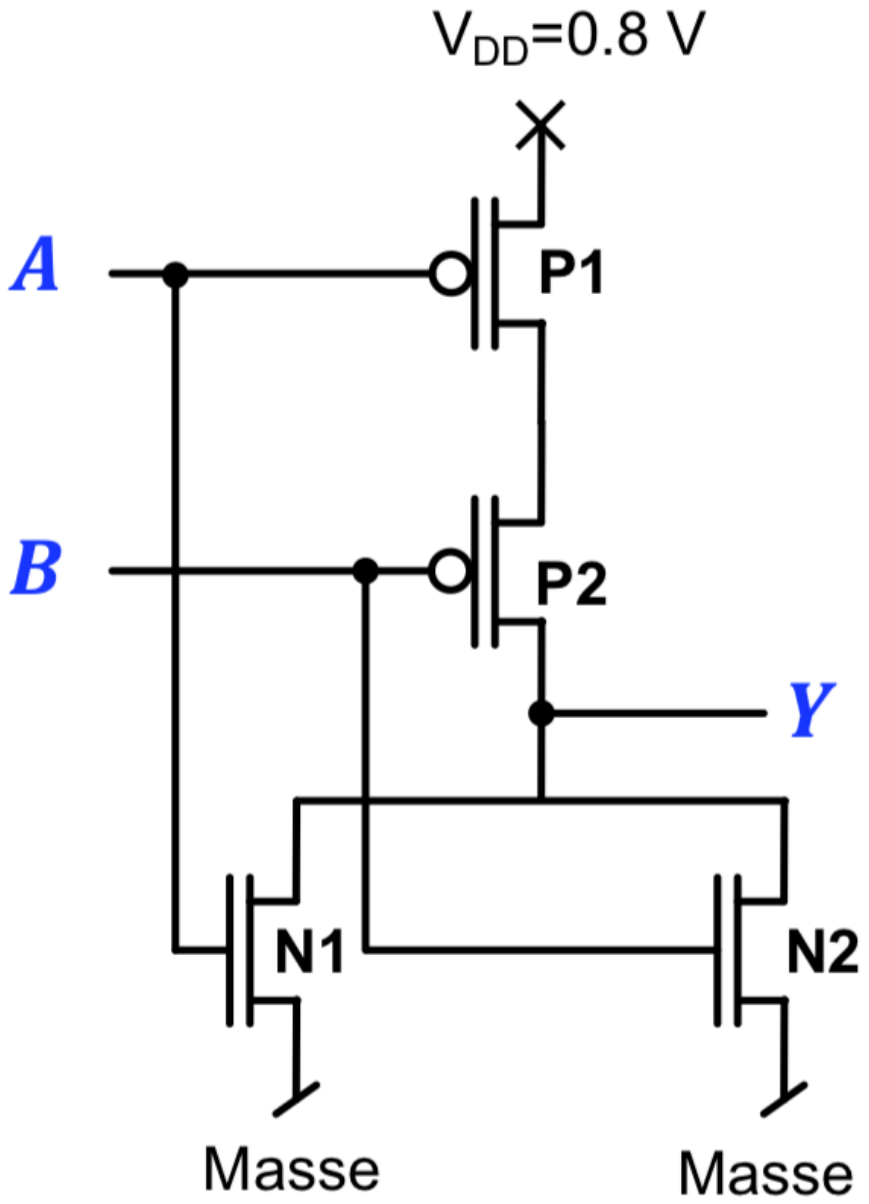
A	PMOS	NMOS	Y
0			
1			

NAND



- PMOS parallel
- NMOS in Serie

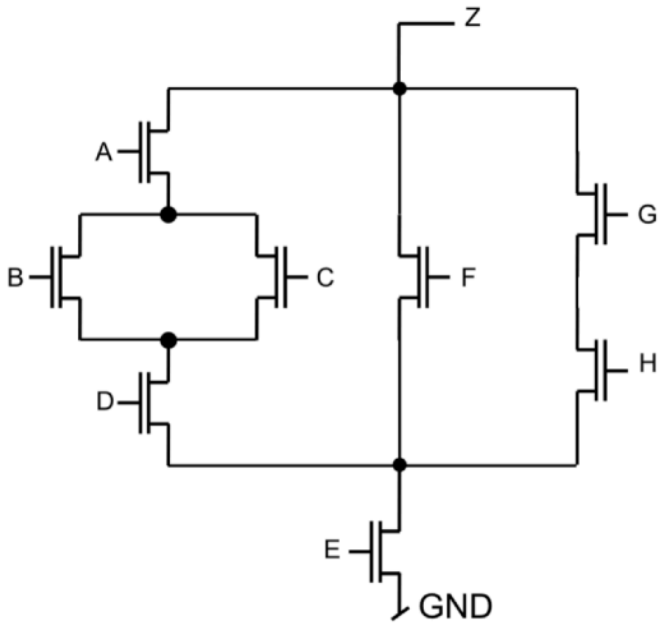
NOR

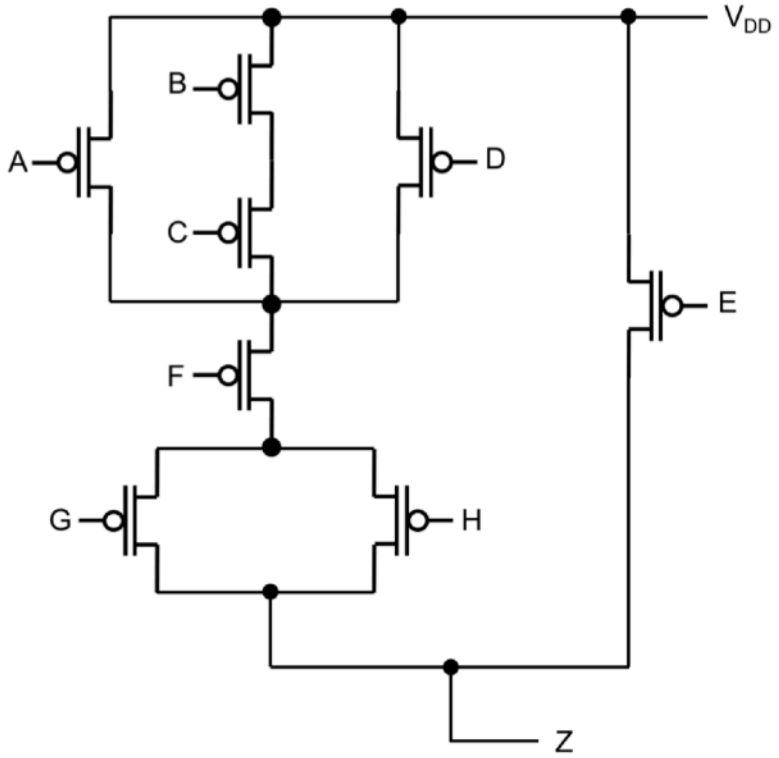


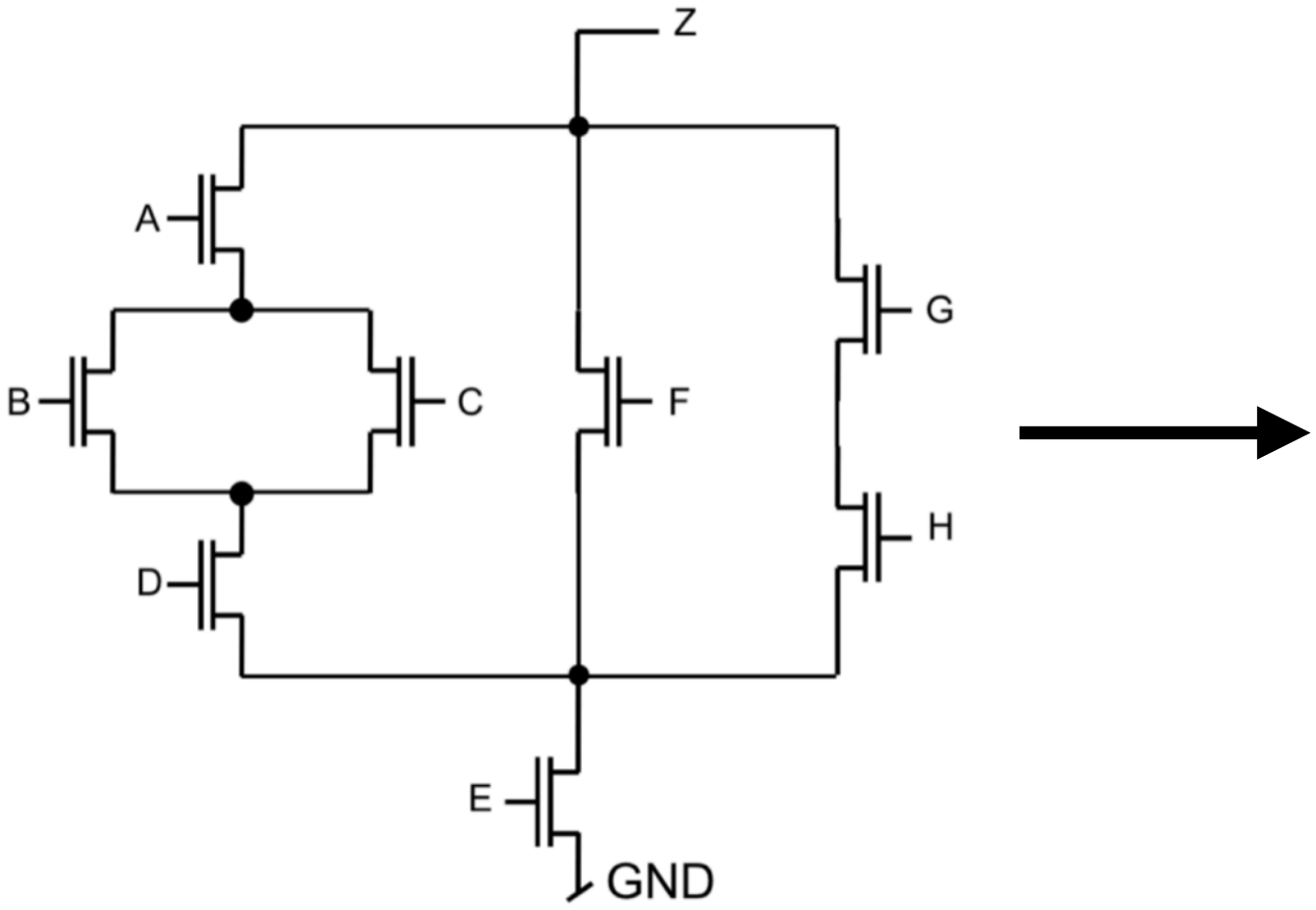
- PMOS in Serie
- NMOS parallel

Pfade umwandeln

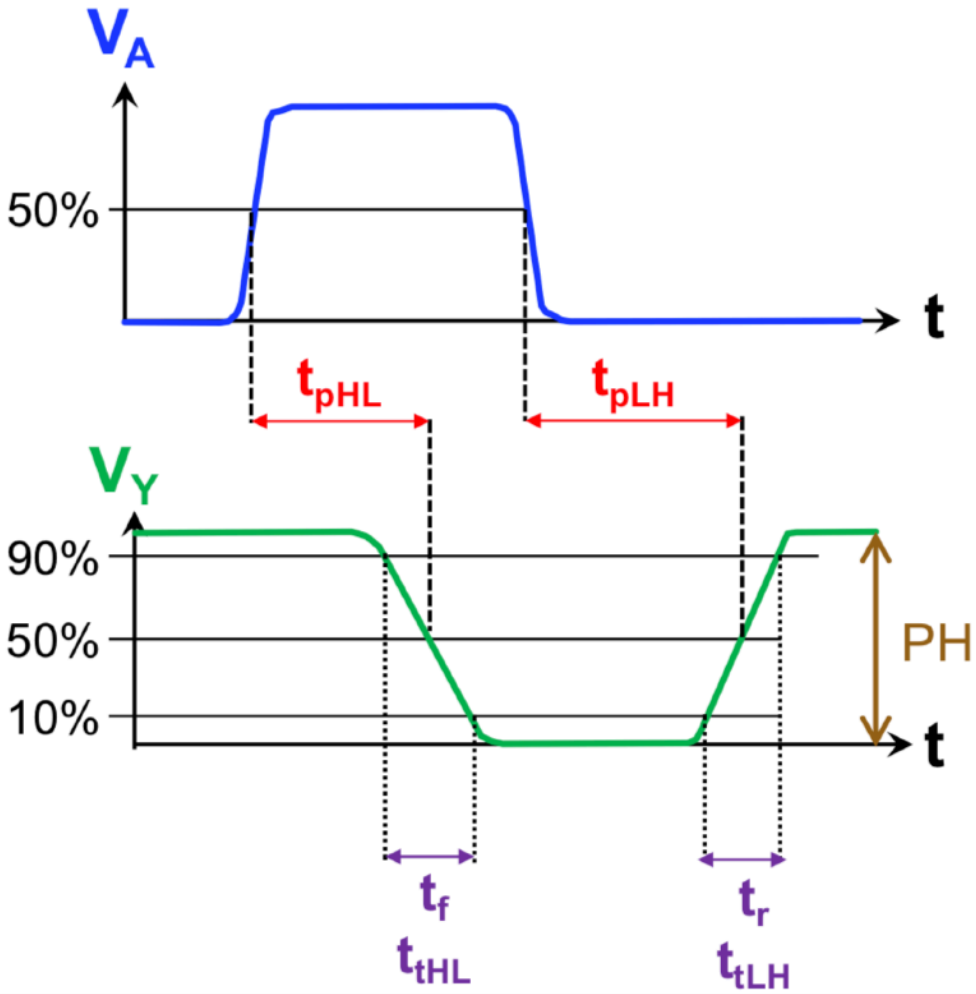
1. Gleichung aufstellen
   - (Bei Pull-up einzelne Elemente invertiert, bei Pull down ganze Gleichung)
   - (UND in Serie, ODER parallel)
2. Gleichung invertieren
   - (UND/ODER vertauschen und Investierungen vertauschen)
3. Neue Gleichung zeichnen
   - (UND: Serie, ODER: Parallel)






Z
A
B
C
D
E
F
G
H
GND

Zeitverzögerung



$$t_d = (t_{pHL}+t_{pLH})/2$$

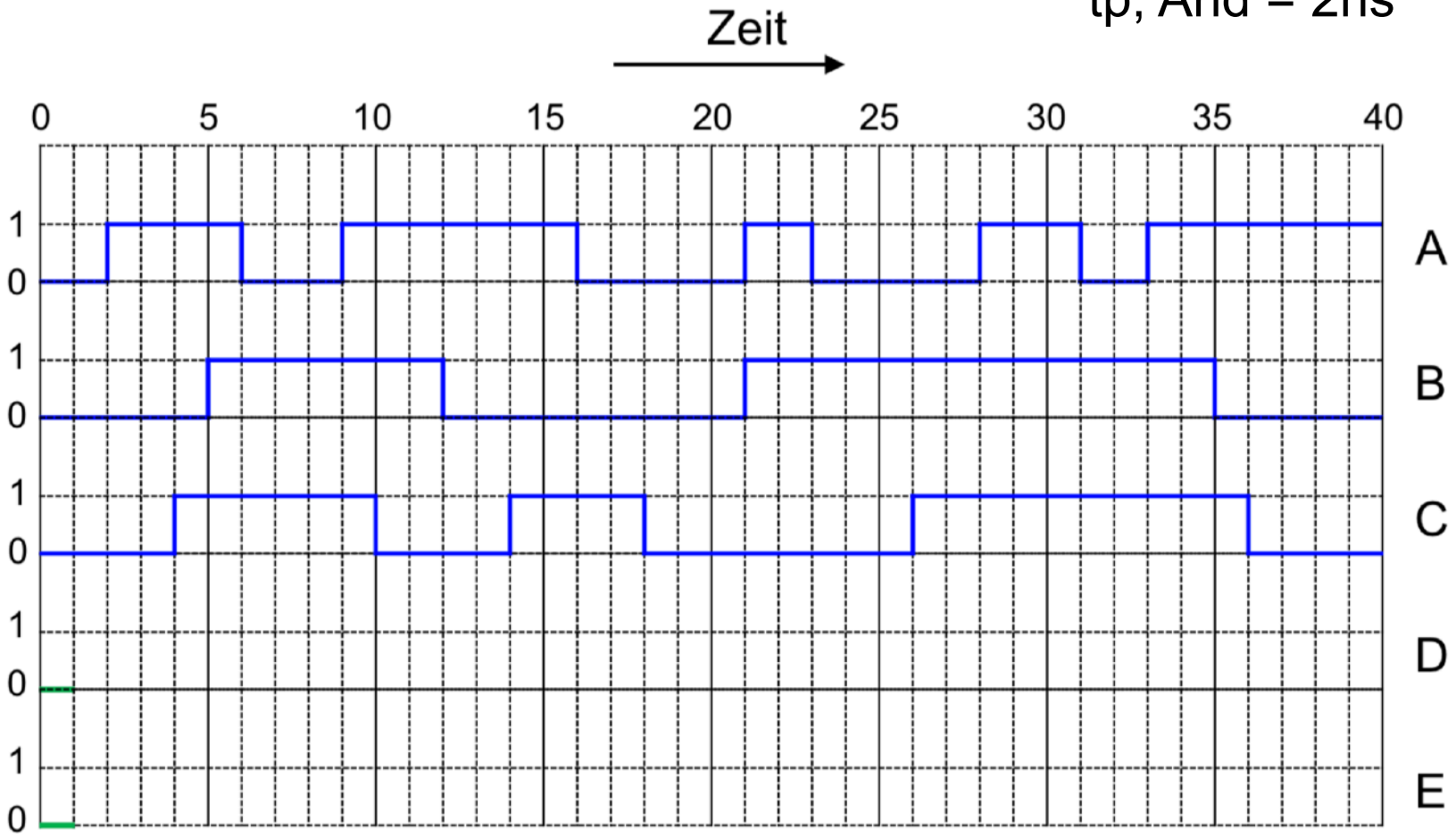
- Verzögerungszeit, gemessen bei 50%
- “Verschiebung im Block”

- Anstiegs- Abfallszeit, gemessen bei 10%-90%
- “Schräge Übergänge”

Zeitverzögerung

D = A * B
tp, And = 2ns

E = D + C
tp, OR = 3ns



Schritt 1: Signal ohne Zeitverzögerung einzeichnen

Bool’sche Algebra

Basisregeln Schaltalgebra

- Kommutativität: (A + B + C = B + C + A) (A * B * C = B * A * C)
- Assoziativität: (A + (B + C) = (A + B) +C) (A * (B * C) = (A * B) * C)
- Distributivität: (A * B + A * C = A * (B + C)) ((A + B) * (A + C) = A + (B * C))

- “Normale” Regeln für Multiplikation und Addition

Bool’sche Grundregeln

NICHT	!!A = A	-
NULL	A + 0 = A	A * 0 = 0
EINS	A + 1 = 1	A * 1 = A
IDEMPOTENZ	A + A = A	A * A = A
KOMPLEMENT	A + !A = 1	A * !A = 0
ADSORPTION	A + (!A * B) = A + B	A * (!A + B) = A * B
ABSORPTION	A + (A * B) = A	A * (A + B) = A
NACHBARSCHAFT	(A * B) + (!A * B) = B	(A + B) * (!A + B) = B

Vorrangsregeln

- Klammern
- {AND; OR; NOR; NAND} vor {XOR; XNOR}
- {AND; OR; NOR; NAND} und {XOR; XNOR} sind untereinander gleichwertig

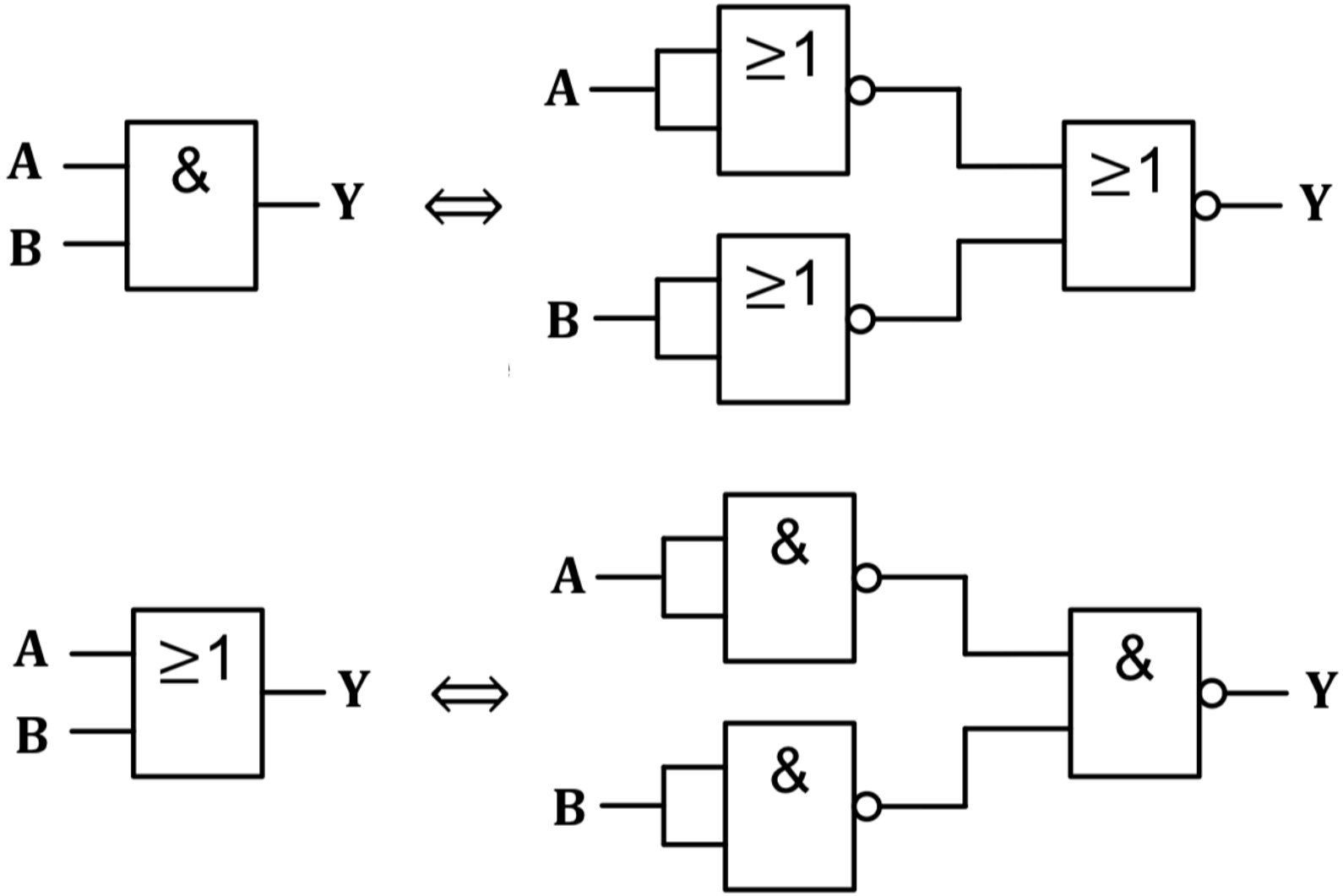
De Morgan’sche Regeln

Die De Morgan‘schen Gesetze sind verallgemeinerbar auf mehreren Variablen:

Erstes Gesetz: $$Y = \overline{A \wedge B \wedge C \wedge \cdots} = \bar{A} \vee \bar{B} \vee \bar{C} \vee \cdots$$

Zweites Gesetz: $$Y = \overline{A \vee B \vee C \vee \cdots} = \bar{A} \wedge \bar{B} \wedge \bar{C} \wedge \cdots$$

De Morgan’sche Regeln



Schaltungssynthese

Minterm

- UND-Verknüpfung von Schaltvariablen
- Minterm gibt nur bei einer Variablenkombination 1 (Minimum)
- Bei n Variablen 2^n Minterme

- Bildung und Variablenfindung:
  - Invertierte = 0
  - Nicht-invertierte = 1

Maxterm

- ODER-Verknüpfung von Schaltvariablen
- Maxterm gibt nur bei einer Variablenkombination 0 (Maximum)
- Bei n Variablen 2^n Maxterme

- Bildung und Variablenfindung:
  - Invertierte = 1
  - Nicht-invertierte = 0

Normalformen

Disjunktive Normalform

- ODER-Verknüpfung von allen Mintermen = 1

Konjunktive Normalform

- UND-Verknüpfung von allen Maxtermen = 0

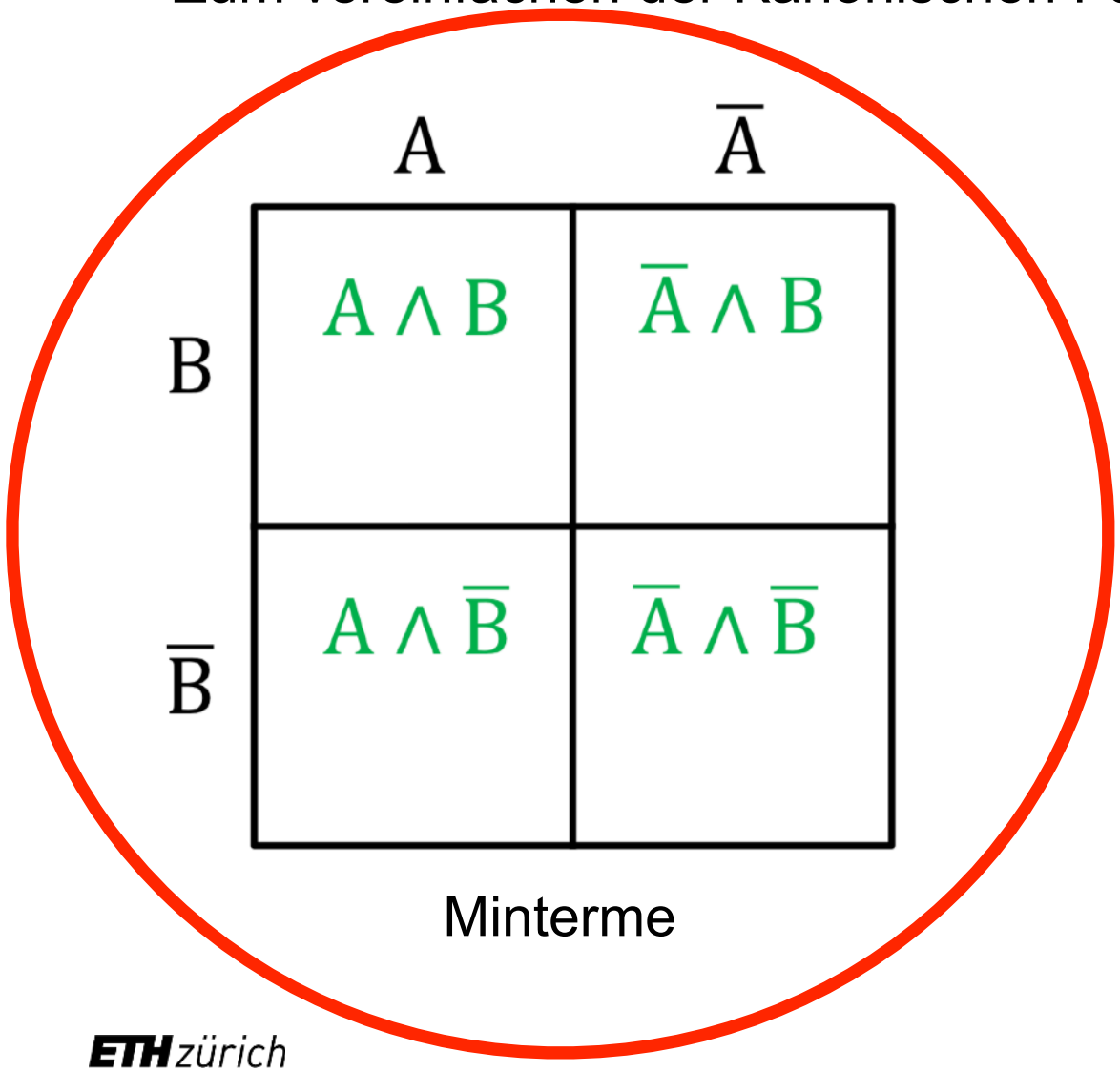
Kanonische Form

- Nur Min- oder Maxterme mit jeder Variable genau 1 Mal

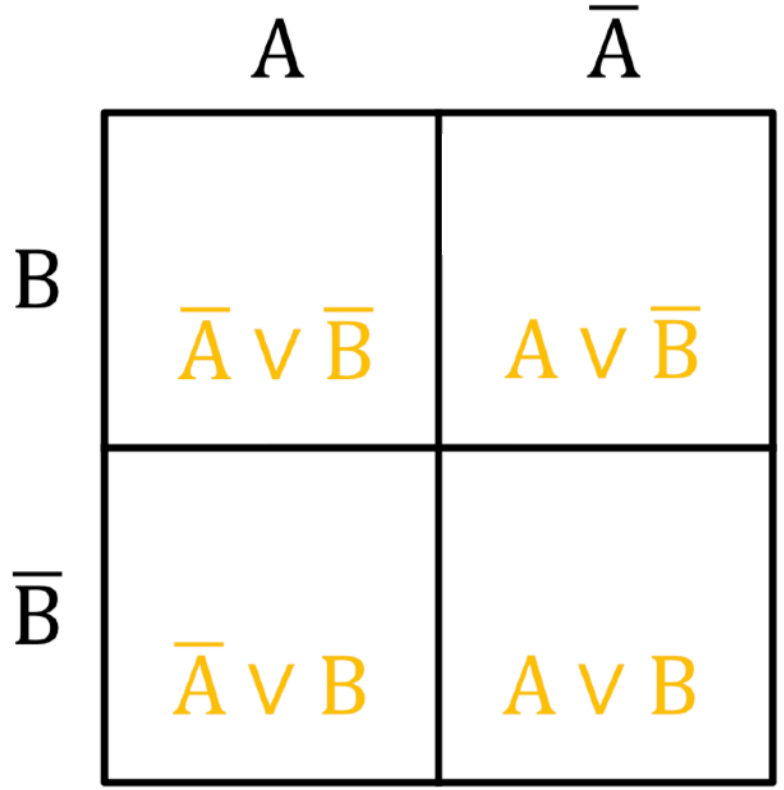
A	B	C	$Y = f(A, B, C)$	Minterm	Maxterm
0	0	0	1		
0	0	1	1		
0	1	0	0		
0	1	1	1		
1	0	0	1		
1	0	1	1		
1	1	0	0		
1	1	1	0		

Karnaugh Diagramme

- Zum vereinfachen der Kanonischen Formen



Minterme



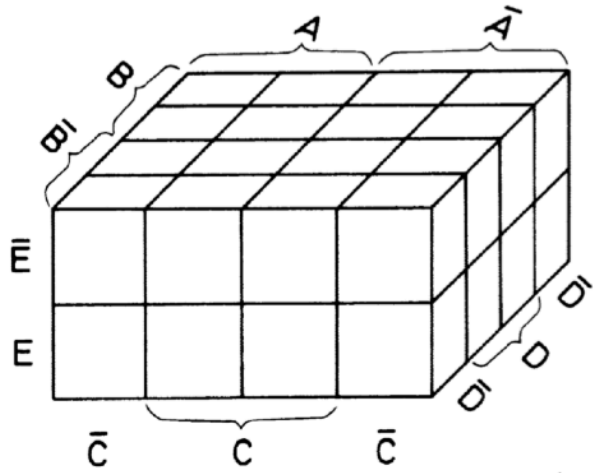
Maxterme

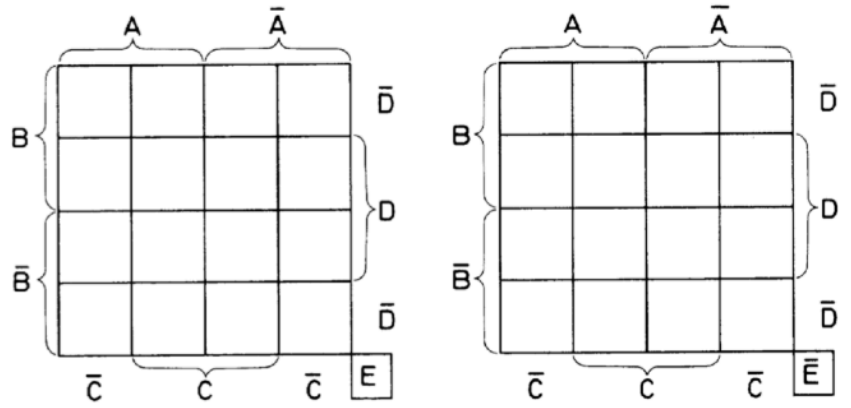
Vereinfachen mit dem Karnaugh Diagramm (DNF)

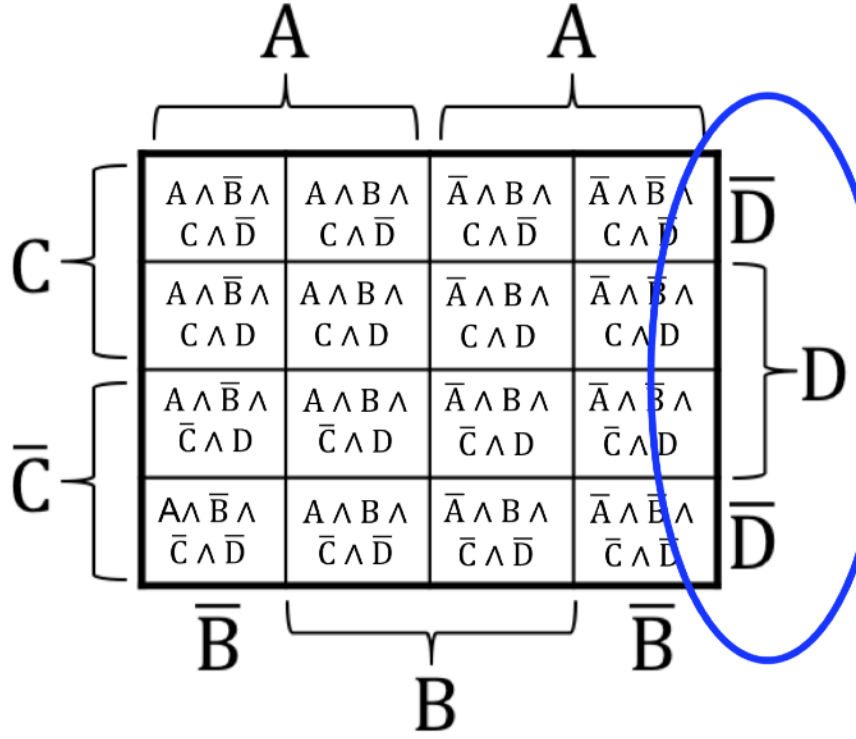
- 1 eintragen, wenn der Minterm existiert.
- 0 eintragen, wenn der Minterm nicht existiert.

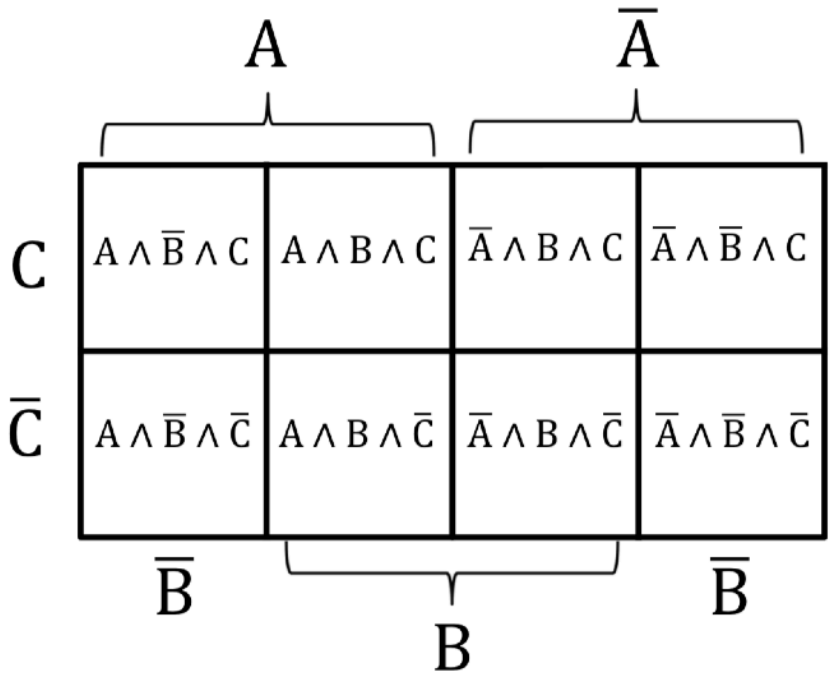
- Päckchen können gebildet werden, wenn:
  - mind. eine Variable negiert und nicht negiert vorkommt
  - mind. eine Variable konstant bleibt
  - Felder könne zu mehreren Päckchen gehören
  - Aufgeschrieben wird die Variable, die konstant bleibt

Mehr Variablen möglich…

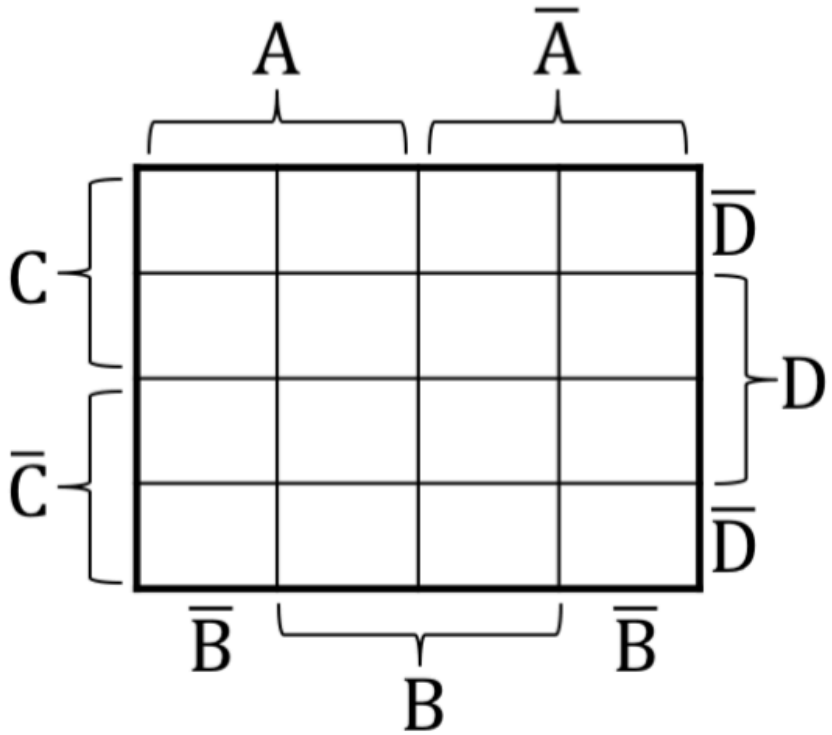








$$f = (\overline{B} \wedge \overline{C}) \vee (A \wedge \overline{B}) \vee (A \wedge B \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge C \wedge \overline{D}) \vee (\overline{A} \wedge \overline{B} \wedge \overline{C} \wedge \overline{D})$$



Vereinfachung mit der KNF

- 0-Päckchen
- Maxterme formen und mit UND verknüpfen
- Eingänge invertieren
- Siehe Serie

Don’t Care Zustände

- Nicht benutzte Zustände können im Karnaugh Diagramm mit X gekennzeichnet werden.
- X können bei der Päckchenbildung mitbenutzt werden.

Hazards

- Durch Zeitverzögerung
- Bei Päckchen, die sich orthogonal berühren
- Behebung: Zusätzliches Päckchen um die betroffenen Stellen

Zahlen & Codes

Zahlensysteme Definition (Das Dezimalsystem)

Basis

$$D = \sum_{i=-\infty}^{i=\infty} b_i \cdot R^i$$

Zwischen 0 und R-1

357 = 7 * 1 + 5 * 10 + 3 * 100 = 3 * 10^0 + 5 * 10^1 + 3 * 10^2

Gebräuchliche Zahlensysteme

- Dezimal
  - Basis = 10
- Binär (0b)
  - Basis = 2
- Hexadezimal (0x)
  - Basis = 16
  - b = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Oktal (0o)
  - Basis = 8

Umwandlung Dezimal zu System mit Basis R

- Ganze Zahl
  - Division mit R
  - Rest eintragen (von Rechts nach Links)
  - Quotient wieder Teilen bis dieser == 0

- Zahl zwischen 0 und 1
  - Mit R Multiplizieren
  - Zahl vor dem Komma eintragen (von Links nach Rechts)
  - Zahl vor dem Komma wegnehmen und wieder Multiplizieren bis ganze Zahl

Beispiel Dezimal zu System mit Basis R

42 = 0b

0o

0x

Umwandlung Binär zu Hex

- Viererpäckchen bilden (von hinten aus)
- Viererpäckchen umwandeln in Dezimal
- Zahlen >9 als Buchstaben schreiben und alles als Hex schreiben

Umwandlung Binär zu Oct

- Dreierpäckchen bilden (von hinten aus)
- Dreierpäckchen umwandeln in Dezimal
- Als Oct schreiben

Beispiel Binär zu Hex

200 = 0b11001000 = 0x

Beispiel Binär zu Oct

200 = 0b11001000 = 0o

Binärzahlen addieren

- “Normales” schriftliches addieren

	1	1	1		1	1				
		0	1	1	0	0	1.	0	1	1
+		1	1	1	0	1	1.	1	0	0
=	1	0	1	0	1	0	0.	1	1	1

Binärzahlen subtrahieren

- Negatives Zweierkomplement zur zu subtrahierenden Zahl bilden
- “Normales” schriftliches addieren

Zweierkomplement

- Um negative Dualzahlen darstellen zu können

	VZ	Betrag
positiv	0	$\boldsymbol{MSB, MSB_{-1}, MSB_{-2}, \cdots, LSB} = \left\lvert \boldsymbol{X_{(2)}} \right\rvert$
negativ	1	$\rightarrow \left\lvert X_{(2)} \right\rvert$ Bitweise invertieren → Eine $1_{(2)}$ muss an der LSB Stelle addiert werden $\overline{\boldsymbol{MSB}}, \overline{\boldsymbol{MSB_{-1}}}, \overline{\boldsymbol{MSB_{-2}}}, \cdots, (\overline{\boldsymbol{LSB}} + \boldsymbol{1_{(2)}})$

- 42 =

(8 Stellen)

Binärzahlen subtrahieren – Beispiel

A=1101.111 B=111000.001

C = A - B =

Codes

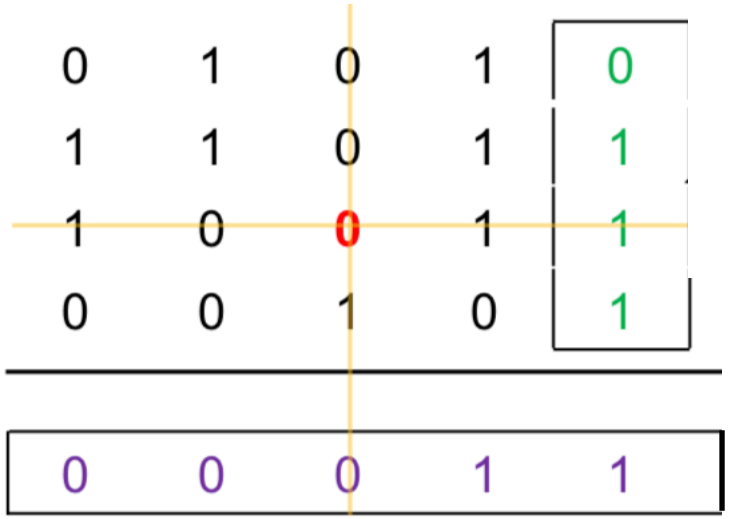
Binär	BCD	Excess-3	Aiken	4-2-2-1	Gray	O'Brien
0000	0		0	0	0	
0001	1		1	1	1	
0010	2		2	2	3	0
0011	3	0	3	3	2	
0100	4	1	4		7	4
0101	5	2			6	3
0110	6	3		4	4	1
0111	7	4		5	5	2
1000	8	5				
1001	9	6				
1010		7				9
1011		8	5			
1100		9	6	6	8	5
1101			7	7	9	6
1110			8	8		7
1111			9	9		8

Einzelne Ziffern in Binärdarstellung

Beim Zählen verändert sich immer nur eine Zahl

Fehlererkennung

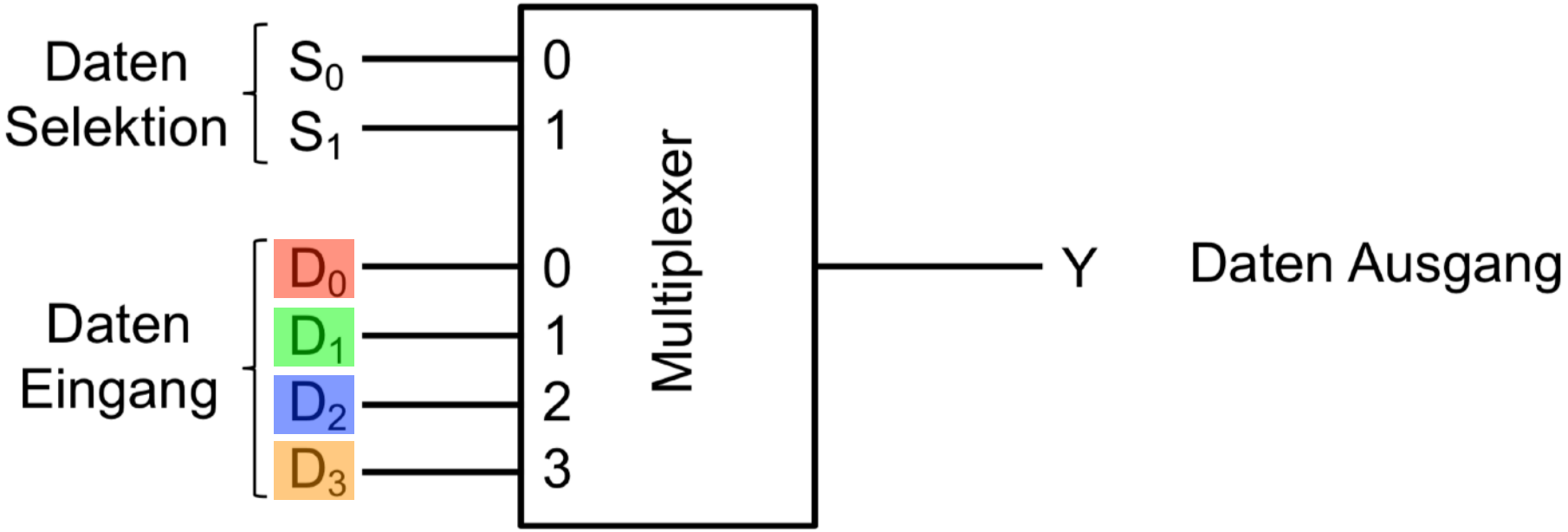
- Code wird in regelmässigen Abständen (Tetrade) um ein Parity Bit ergänzt
  - Even Parity Bit = 0 wenn Anzahl 1 gerade
  - Odd Parity Bit = 0 wenn Anzahl 1 ungerade
- Am Ende des übertragenen Codes wird ein Prüfwort angehängt



Rechenschaltungen

Multiplexer

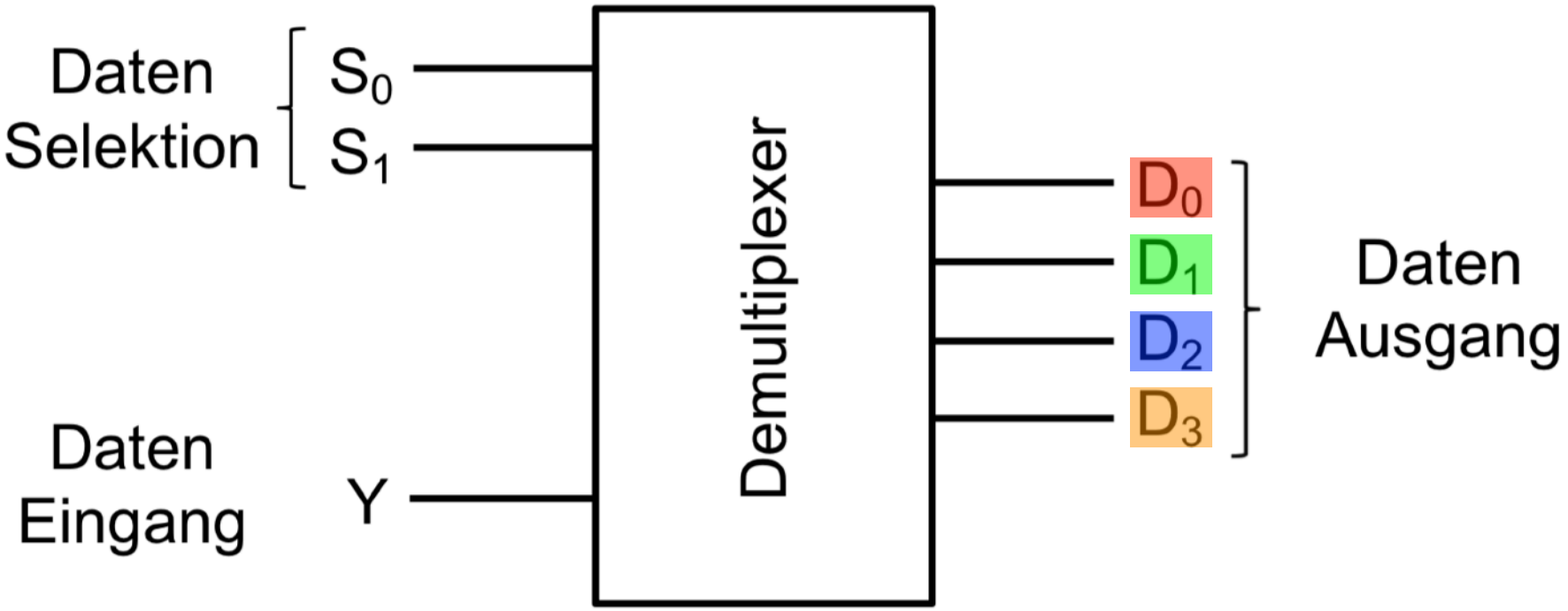
Mehrere Eingänge auf einen Ausgang



$$Y = (\overline{S_0} \wedge \overline{S_1} \wedge D_0) \vee (\overline{S_0} \wedge S_1 \wedge D_1) \vee (S_0 \wedge \overline{S_1} \wedge D_2) \vee (S_0 \wedge S_1 \wedge D_3)$$

Demultiplexer

Ein Eingang auf mehrere Ausgänge

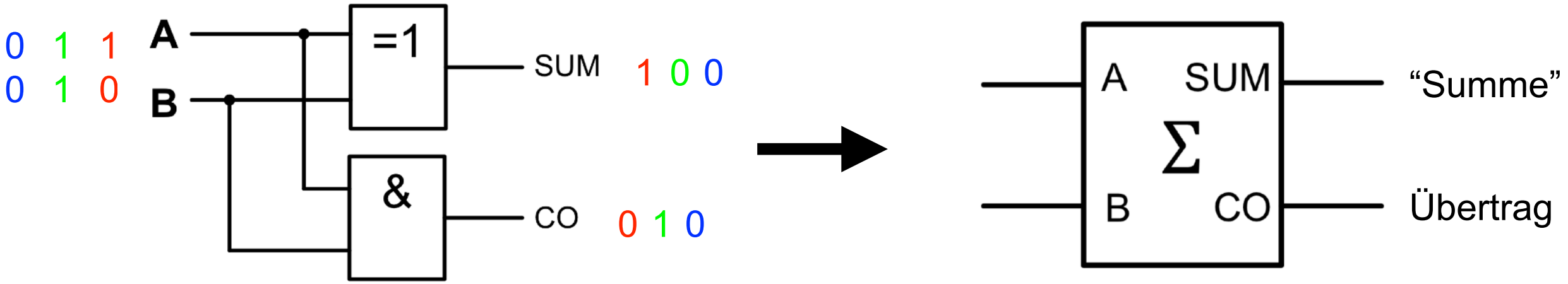


$$D_0 = \overline{S_0} \wedge \overline{S_1} \wedge Y, \qquad D_1 = \overline{S_0} \wedge S_1 \wedge Y,$$

$$D_2 = S_0 \wedge \overline{S_1} \wedge Y, \qquad D_3 = S_0 \wedge S_1 \wedge Y$$

Halbaddierer

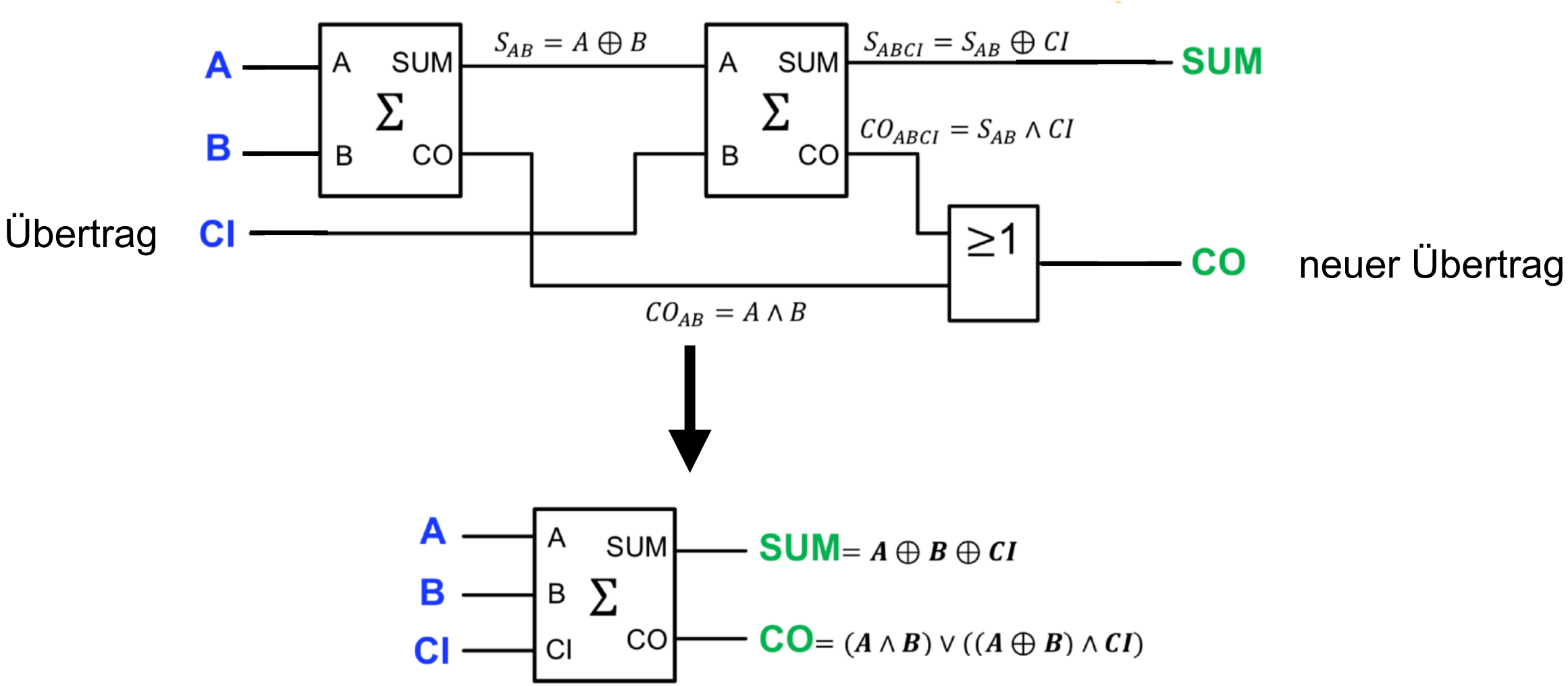
- Um 2 Bits zu addieren



Was machen wir, wenn wir 3 Bits addieren wollen?

Volladdierer

- Um 3 Bits zu addieren
- A + B + C = (A + B) + C



Mehrbit-Addierer

- Wir wollen ganze Dualzahlen addieren

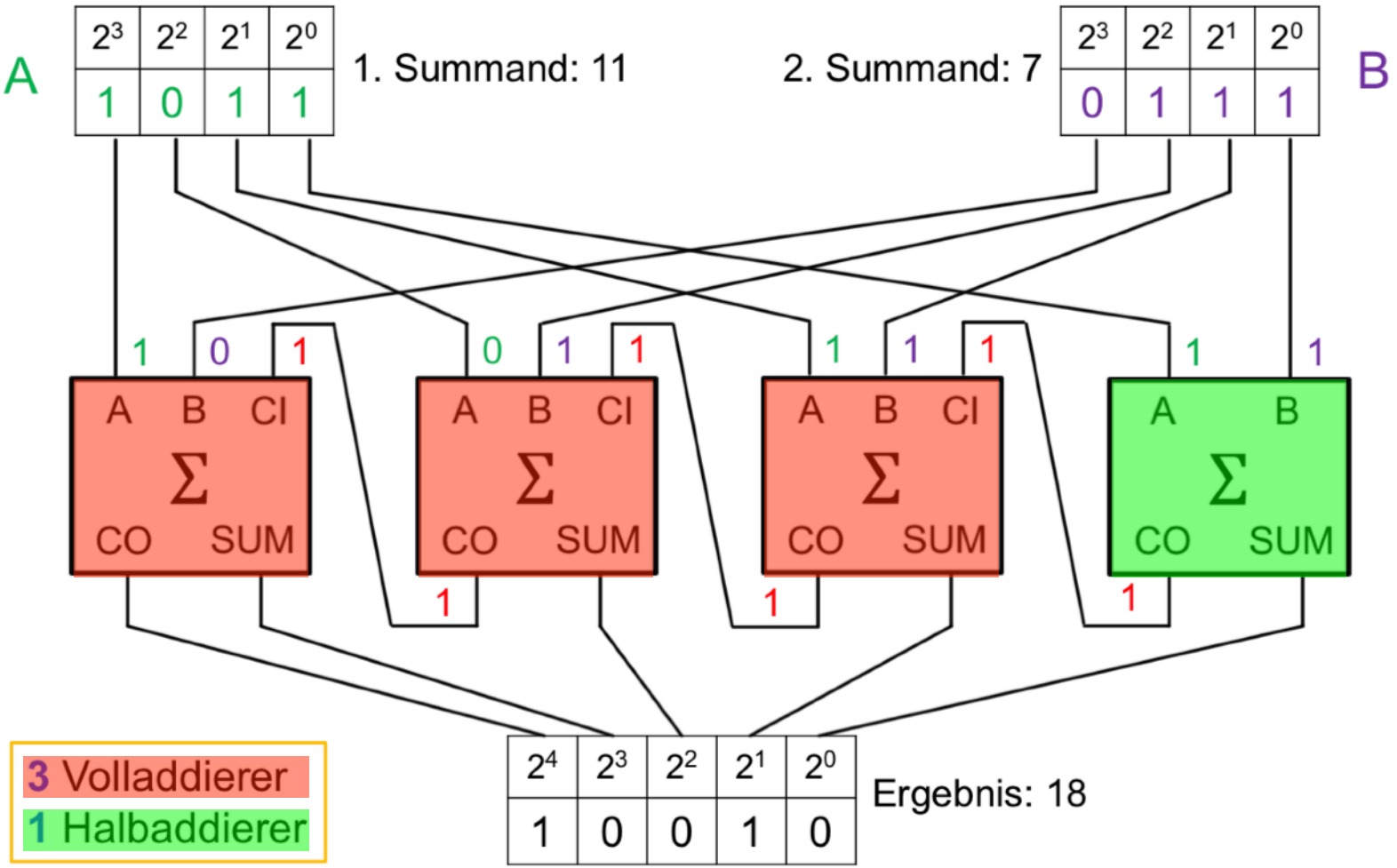
- Serienaddierer: in jedem Taktschritt wird eine Stelle addiert
- Paralleladdierer: in einem Taktschritt werden alle Stellen addiert

Paralleladdierer

- Für jedes Bit eine Schaltung generieren
  - S_0 = A_0 + B_0
  - S_1 = A_1 + B_1 + C0_0

- Vorteile
  - Schnell, da die Daten max. durch 3 Gatter müssen
- Nachteile
  - sehr Schaltungsaufwendig

Ripple-Carry Addierer



Ripple-Carry Addierer II

- Vorteile
  - Schaltungsaufwand wächst linear
  - Einfach erweiterbar

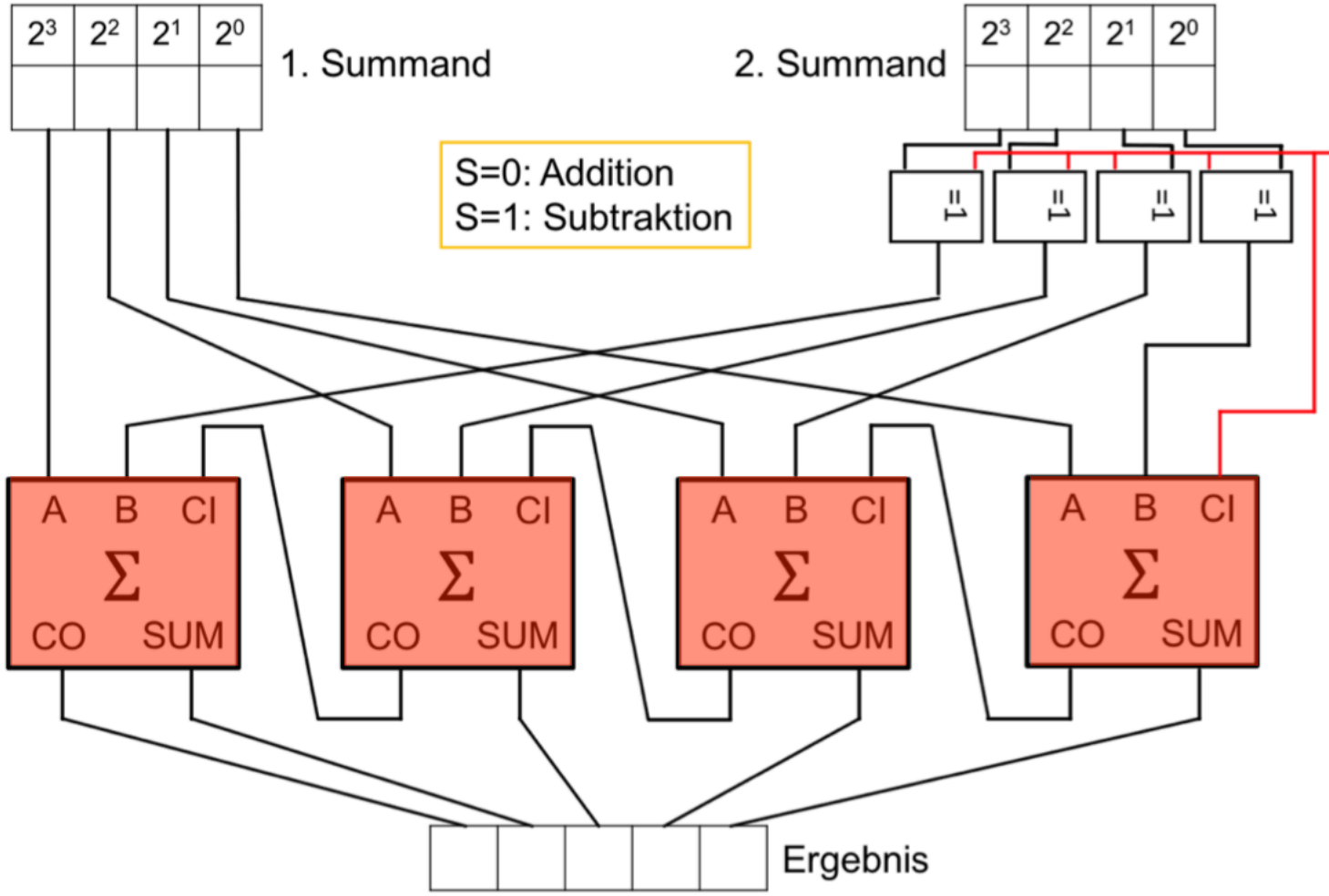
- Nachteile
  - langsam, da die Daten durchrieseln müssen

Carry-Lookahead Addierer

- Addierer werden kaskadiert
- Überträge werden parallel berechnet

- Vorteile
  - Gleich schnell wie der Paralleladdierer
  - Schaltungsaufwand wächst linear (wie bei Ripple-Carry)

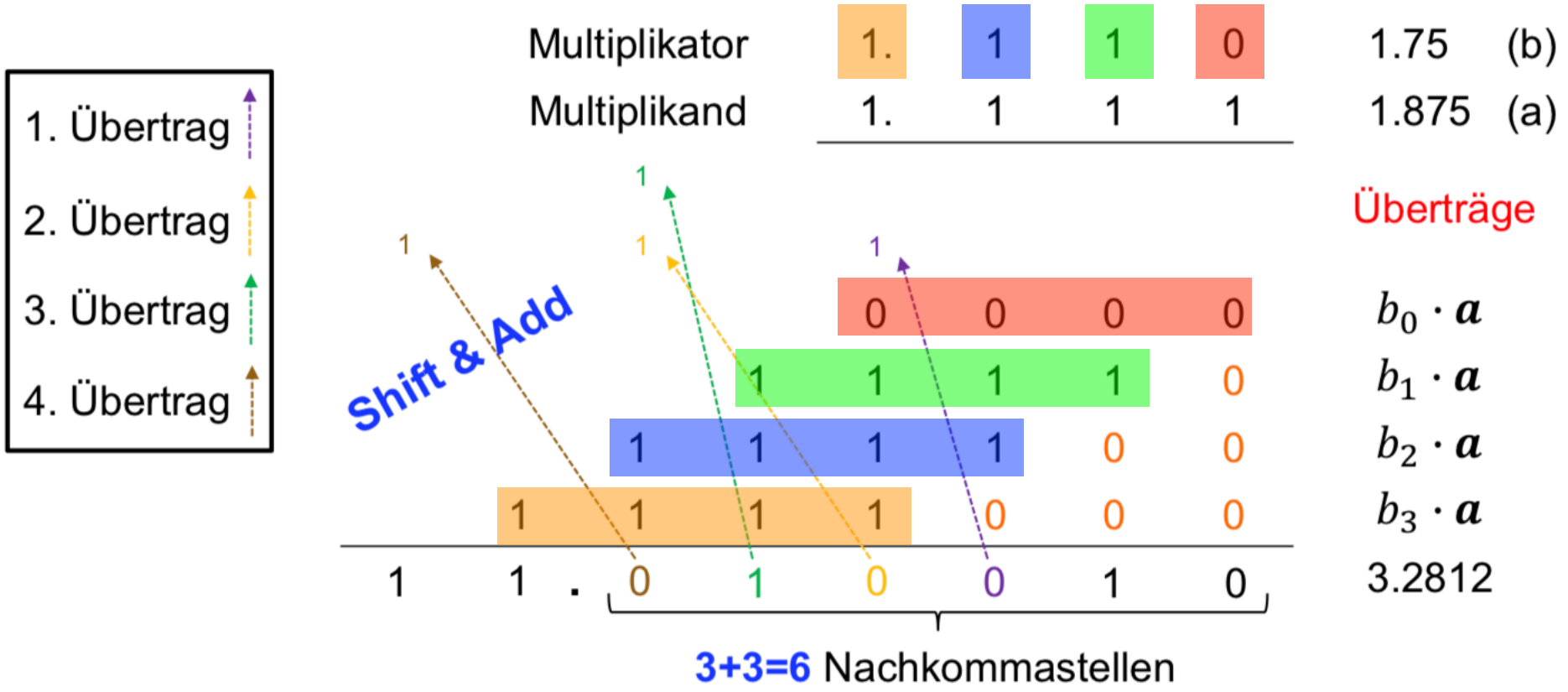
Subtrahierer



S = 0: Addition

S = 1: Subtraktion

Multiplikation von Binärzahlen



Booth-Algorithmus

(0 0): nichts machen

(1 0): -B

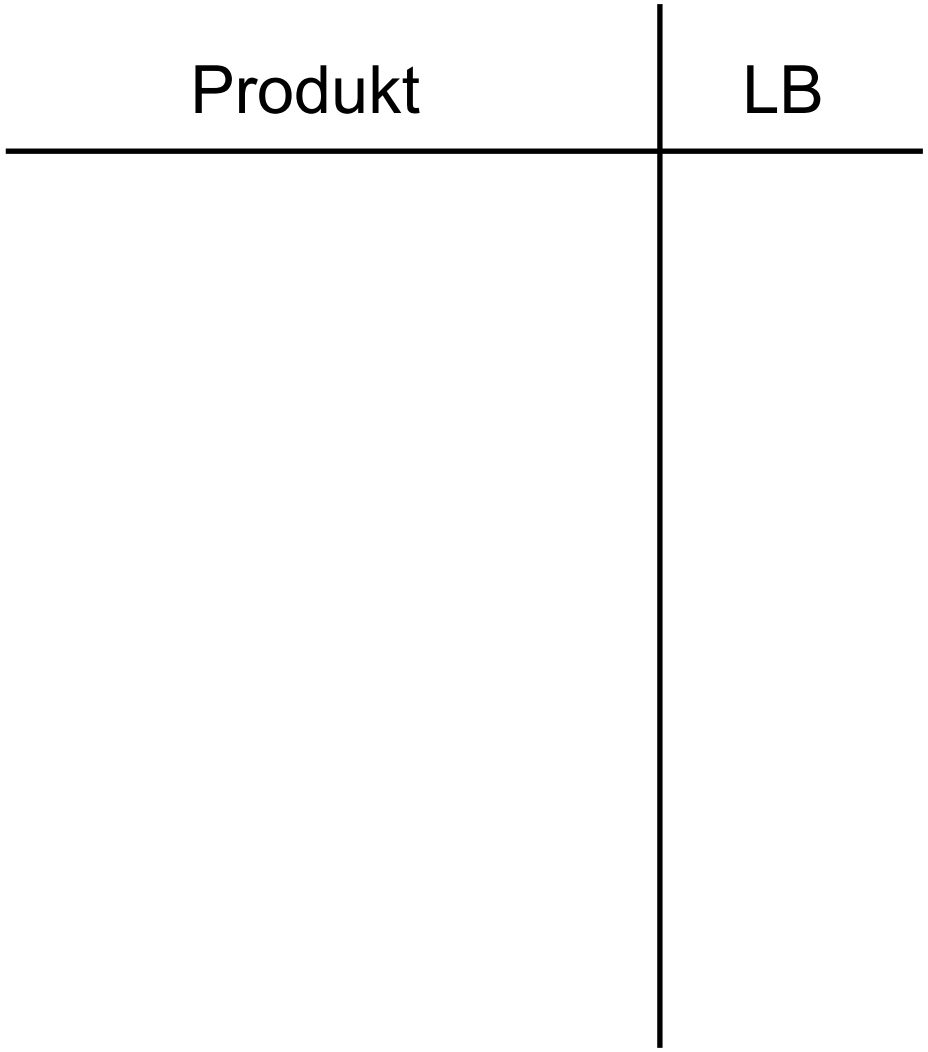
(0 1): +B

Shift

A = 0100

B = 0111

-B = 1001



Latches & Flipflops

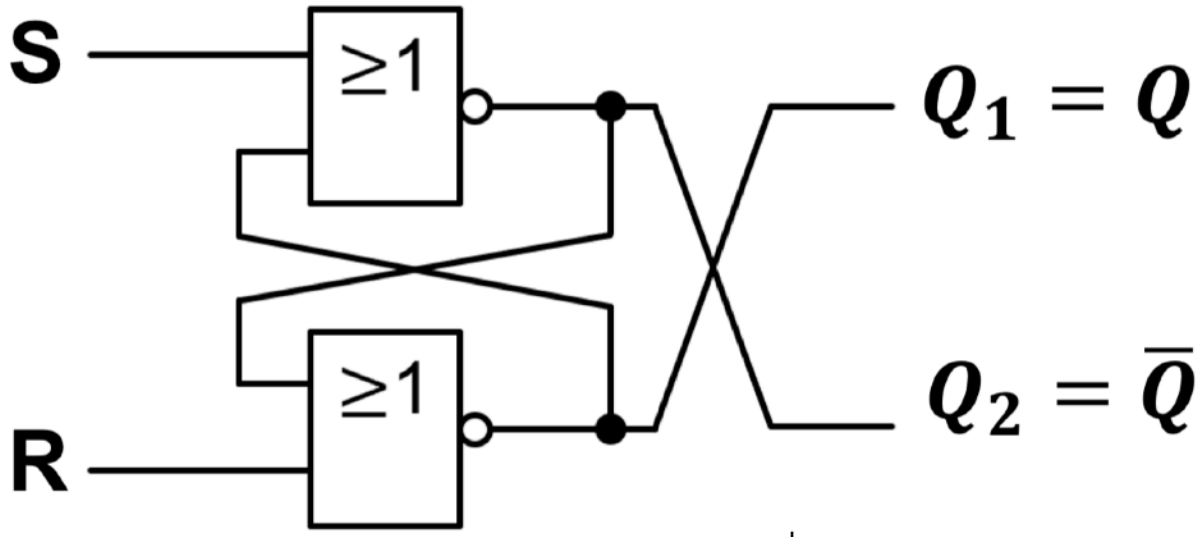
Kombinatorische Schaltungen

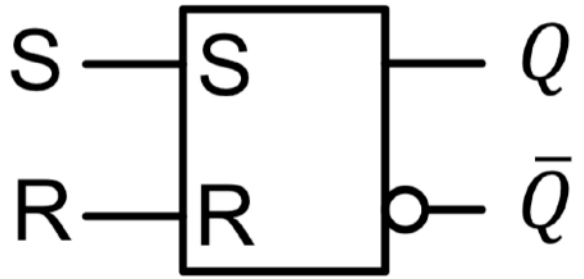
- Ausgang hängt nur von den Eingängen und Gattern ab
- Bisher nur solche betrachtet

Sequentielle Schaltungen

- Besitzen Rückkoppelungen
- jetzt neu

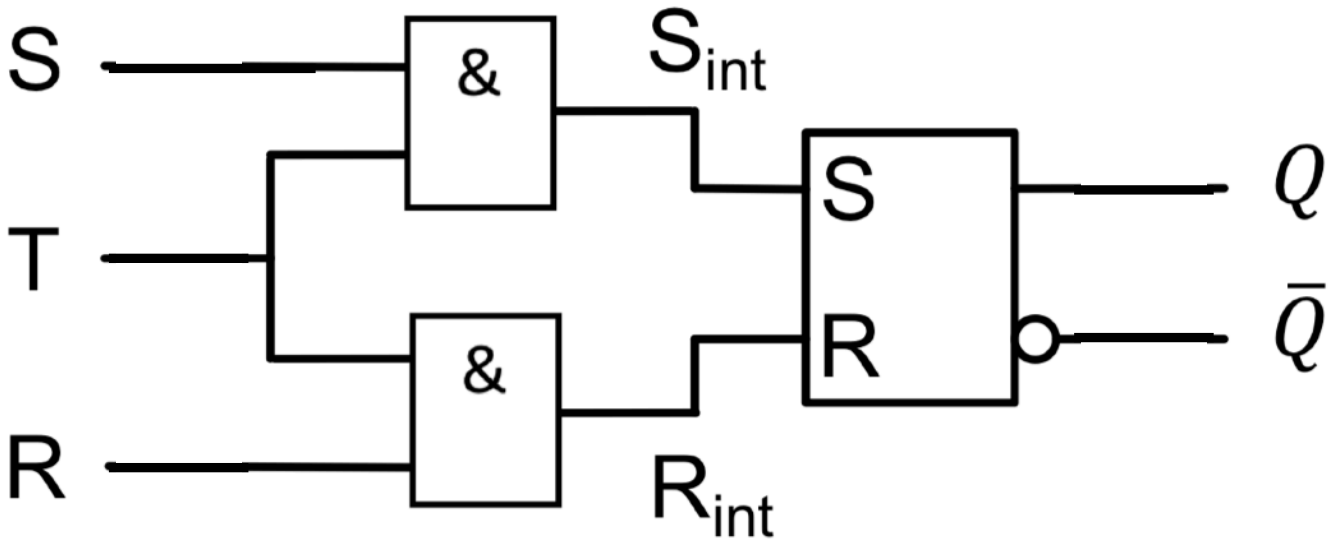
SR-Latch

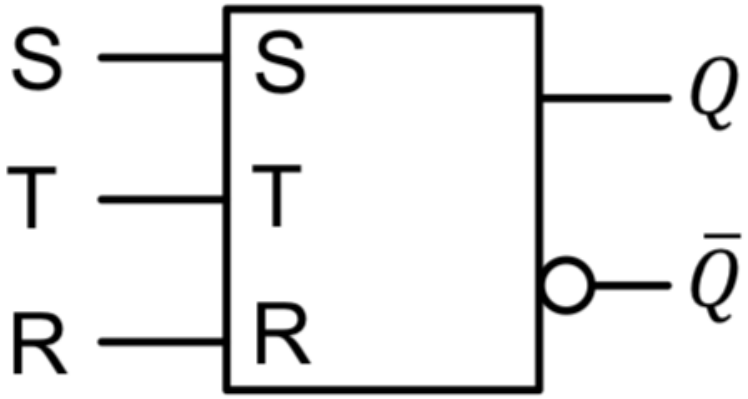




Fall	S	R	Q_{1n}	Q_{1n+1}	
1	0	0	0	0	speichern
2	0	0	1	1	
3	0	1	0	0	rücksetzen
4	0	1	1	0	
5	1	0	0	1	setzen
6	1	0	1	1	
7	1	1	0	X	unzulässig
8	1	1	1	X	

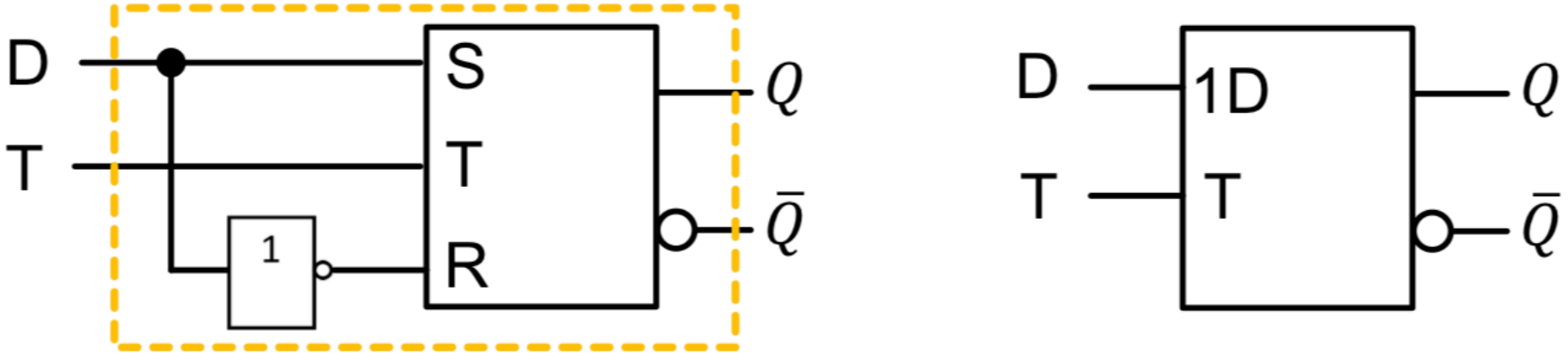
Taktzustandgesteuertes SR-Latch





Gleich wie SR-Latches, nur dass sich Q nur verändern kann, wenn T = 1

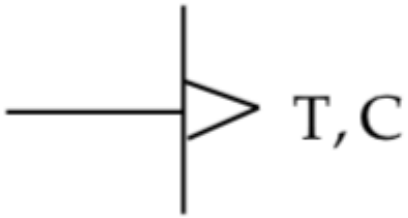
D-Latches



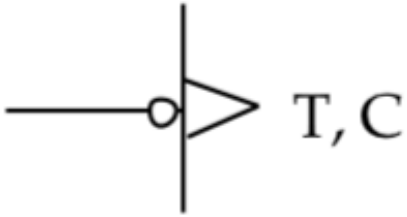
- Die Eingangskombination R = S = 1 wird verhindert
- Q = D während T = 1
- Bei T = 0 wird der letzte Wert vor dem Übergang beibehalten

Flipflops

- TaktFLANKENgesteuert

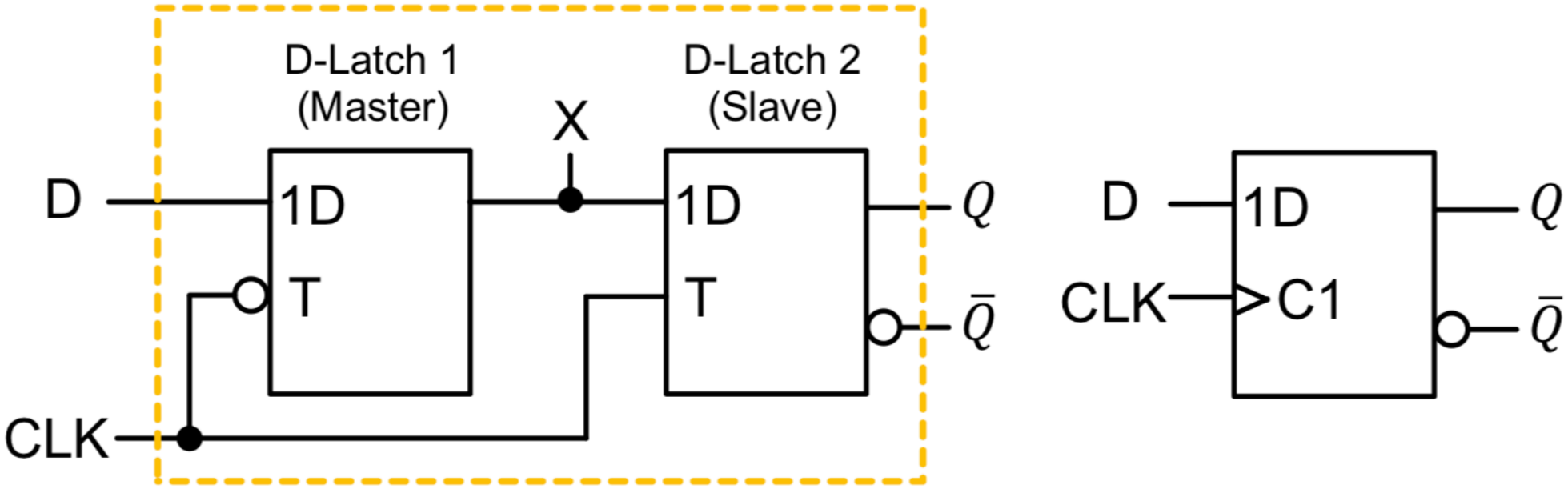


Eingangsvariable werden beim
0 - 1 Übergang von C wirksam



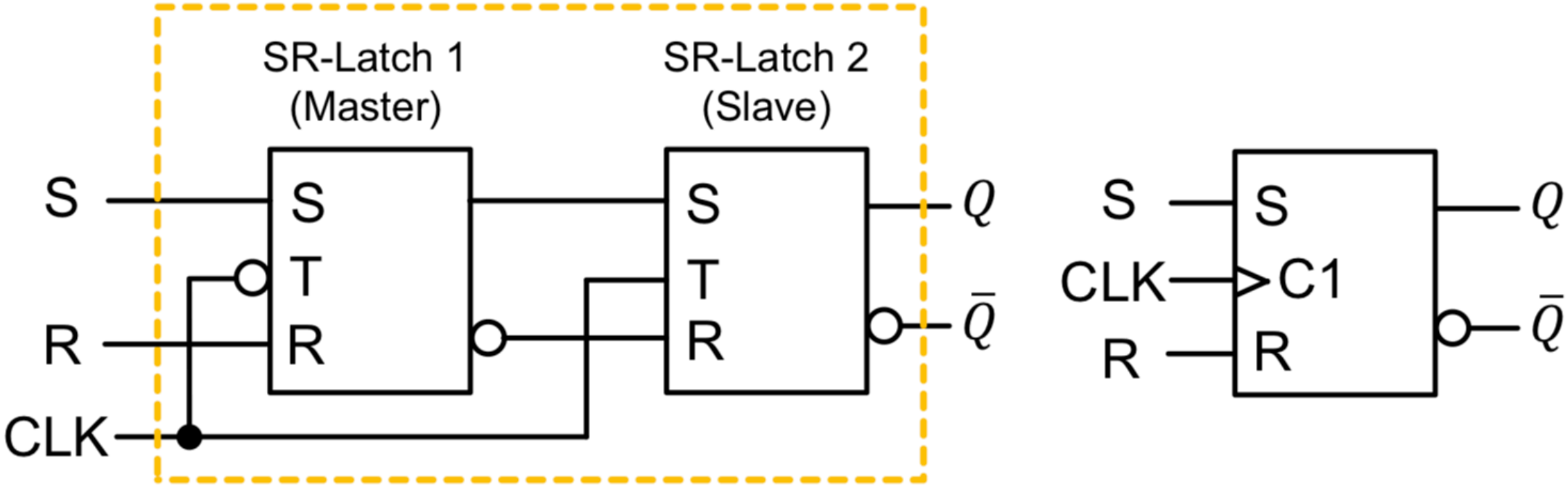
Eingangsvariable werden beim
1 - 0 Übergang von C wirksam

D-Flipflop

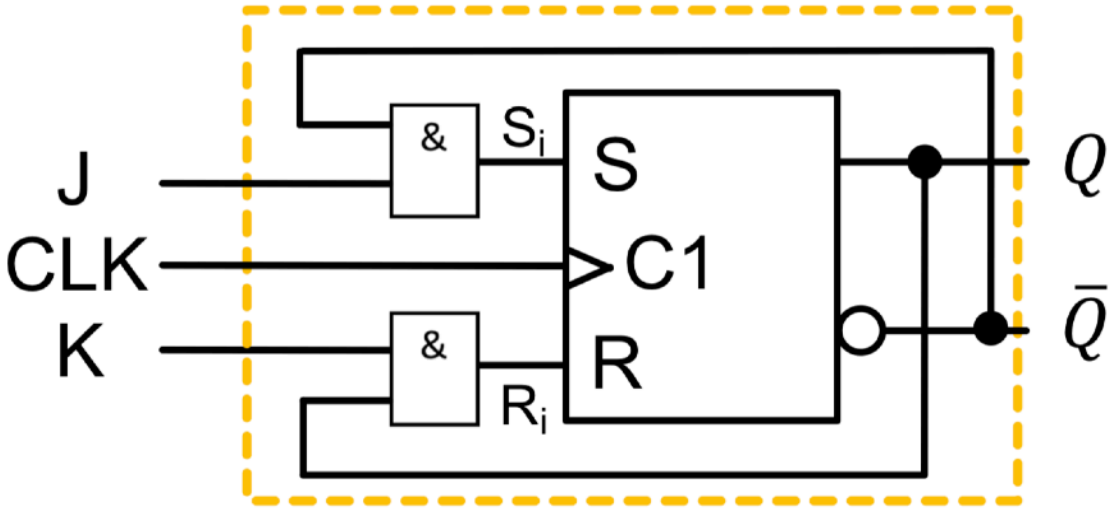


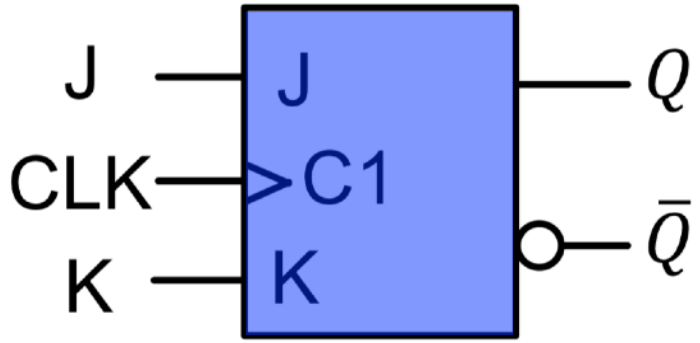
Ausgang wird zu D bei jeder positiven Taktflanke

SR-Flipflop


SR-Latch 1
(Master)
SR-Latch 2
(Slave)
S
R
CLK
S
T
R
S
T
R
Q
$\bar{Q}$
S
CLK
R
S
C1
R
Q
$\bar{Q}$

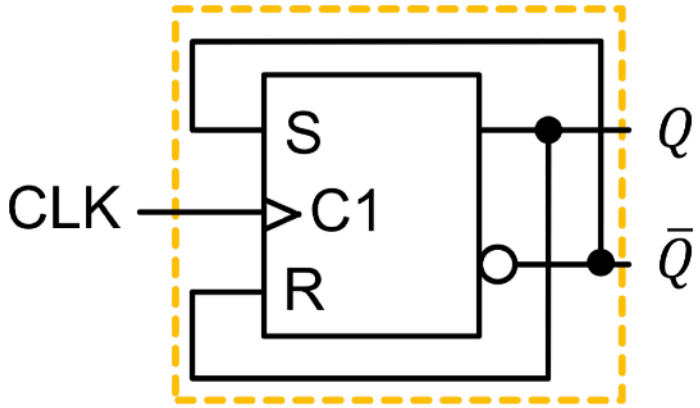
JK-Flipflop (Jump - Kill)



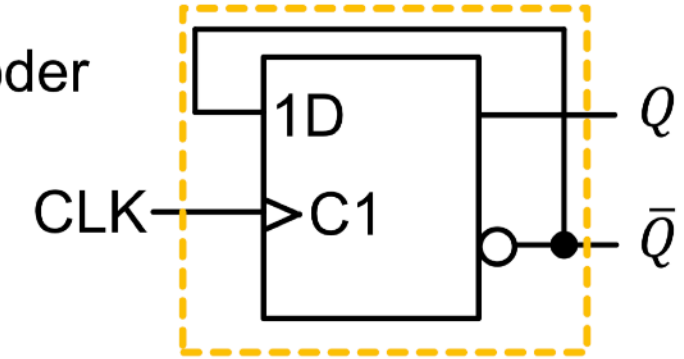


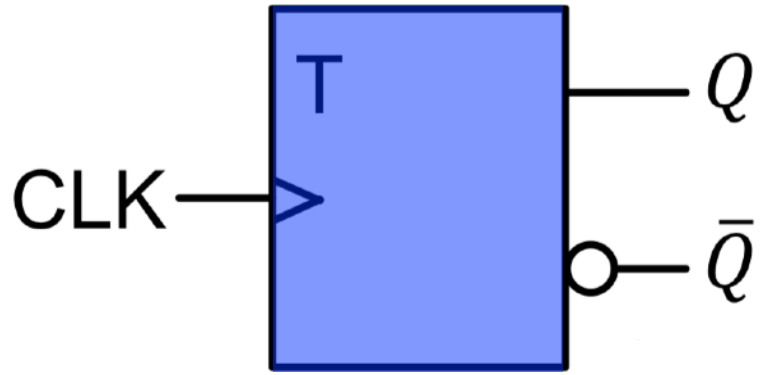
Fall	J	K	Q_{1n+1}	Q_{2n+1}	
1	0	0	Q_{1n}	Q_{2n}	speichern
2	0	1	0	1	rücksetzen
3	1	0	1	0	setzen
4	1	1	$\overline{Q_{1n}}$	$\overline{Q_{2n}}$	wechseln

T-Flipflop (Toggle)



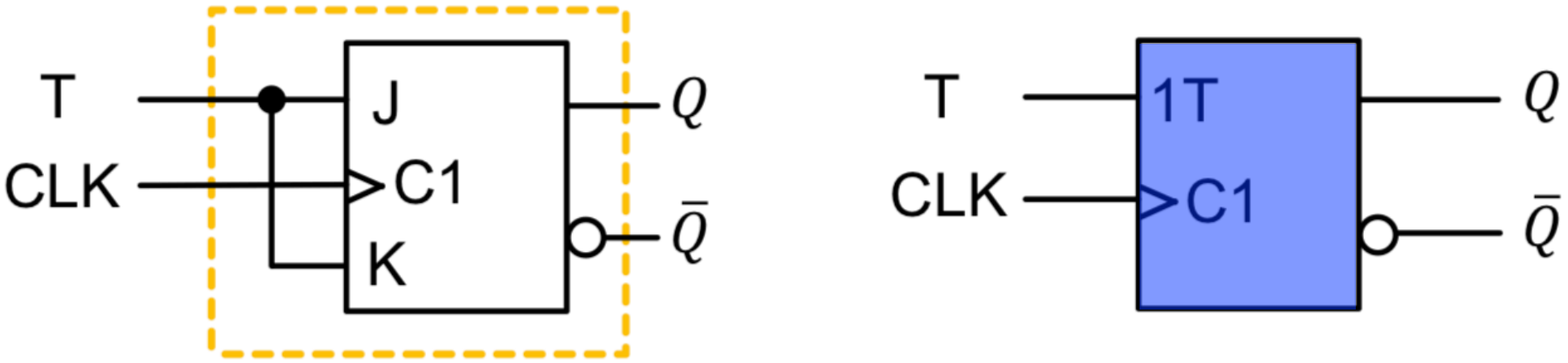
oder





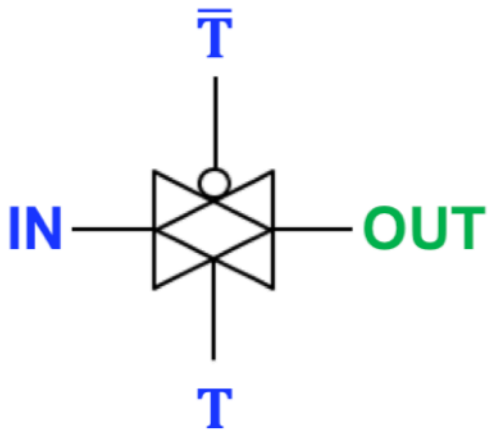
Signal “toggelt” (= wechselt) bei jeder Taktflanke

T-Flipflop mit 2 Eingängen

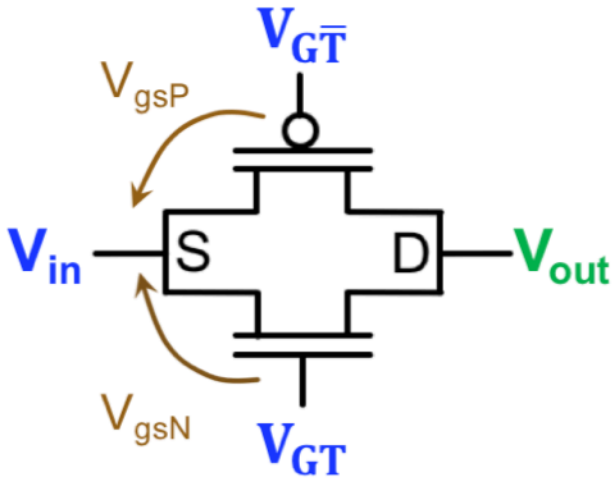


Signal “toggelt” nur, wenn T = 1

Transmission Gate



Schaltsymbol



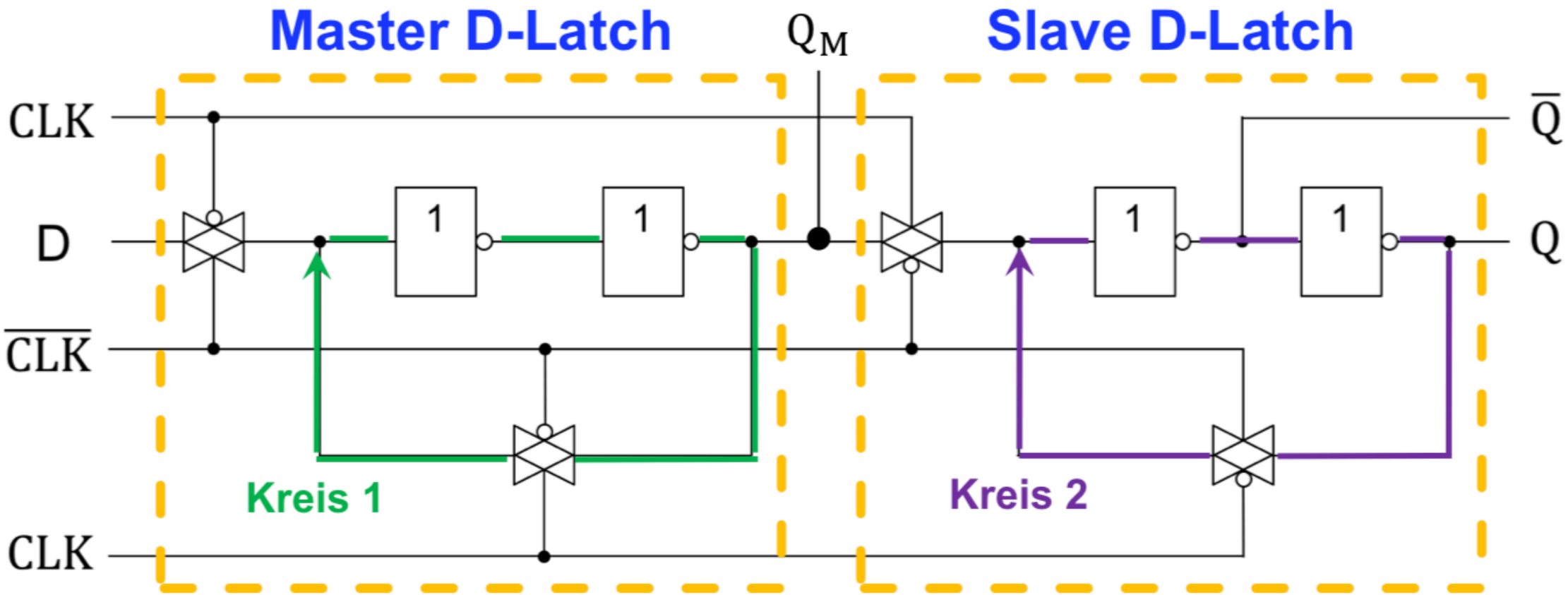
CMOS Schaltung

IN	T	Widerstand	OUT
0	0	hochohmig	-
0	1	niederohmig	0
1	0	hochohmig	-
1	1	niederohmig	1

PMOS leitet, wenn Vs = 1 und Vg = 0

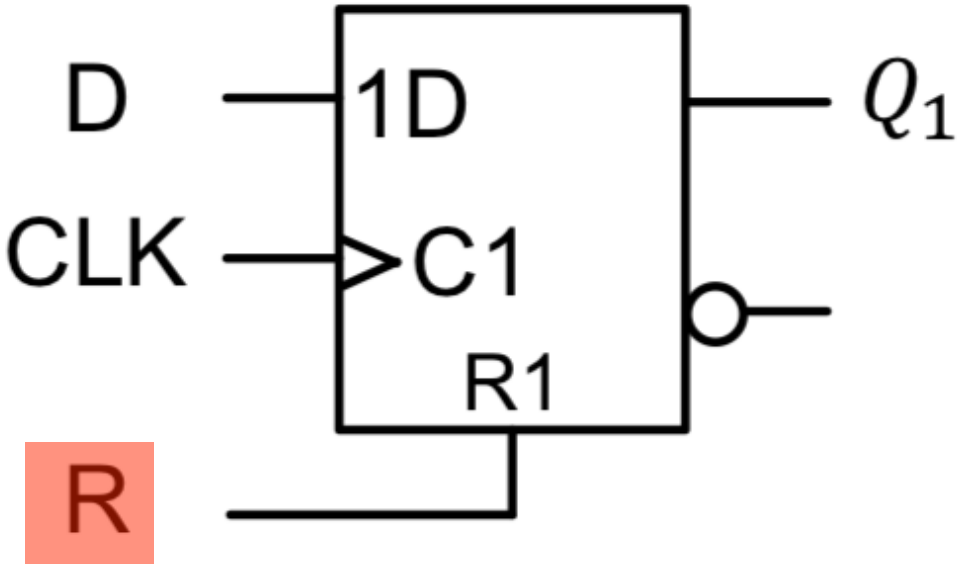
NMOS leitet, wenn Vs = 0 und Vg = 1

D-Flipflop mit Transmission Gates



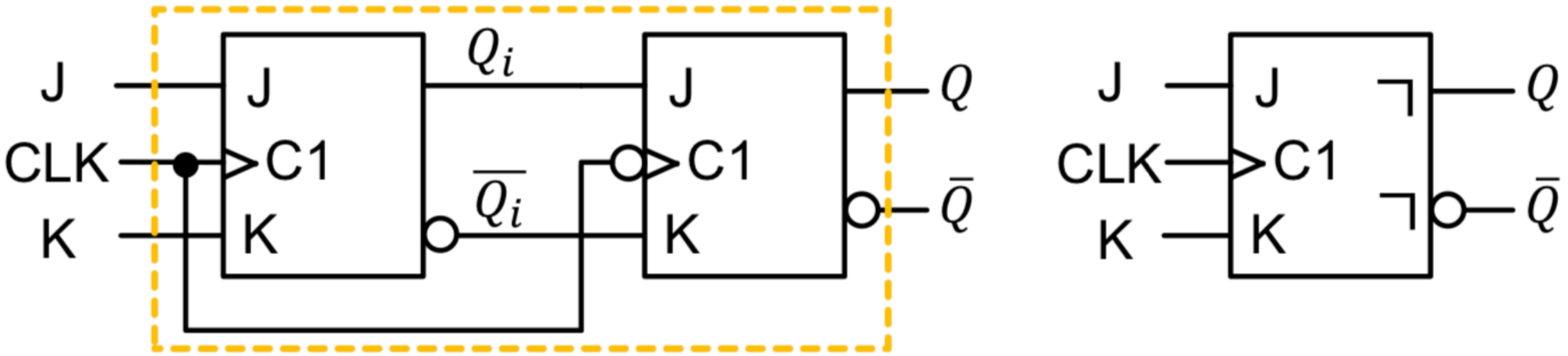
8 NMOS und 8 PMOS

Rücksetz-Eingang



Speicherzustand kann jederzeit auf 0 zurückgesetzt werden

Master-Slave Flipflop



Informationen werden bei steigender Taktflanke eingelesen aber erst bei fallender ausgegeben

Maximale Taktfrequenz

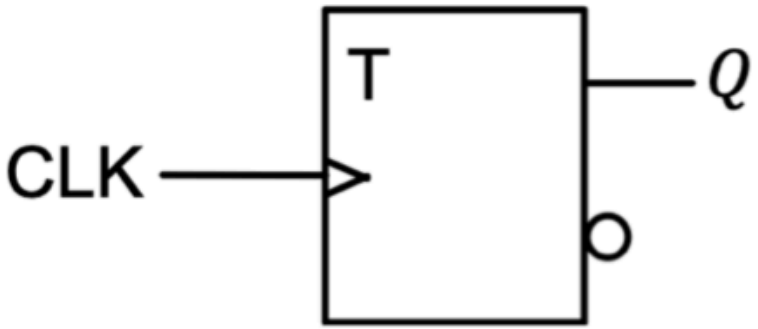
- Wenn mindestens 2 Flipflops in Serie
- t_pd = Verzögerungszeit (Durchlaufzeit)
- t_s = Setup-Zeit (Wie lange ein Signal vor einer Taktflanke unverändert sein muss)
- t_h = Hold-Zeit (Wie lange ein Signal nach einer Taktflanke unverändert sein muss)

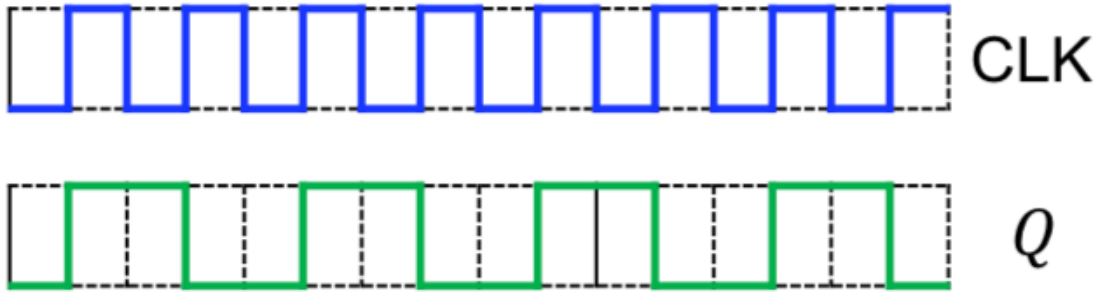
$$T_{min} - t_{s,ff2} \geq t_{pd,ff1} + t_{pd,ks}$$

$$f_{max} = 1/T_{min}$$

Frequenzteiler

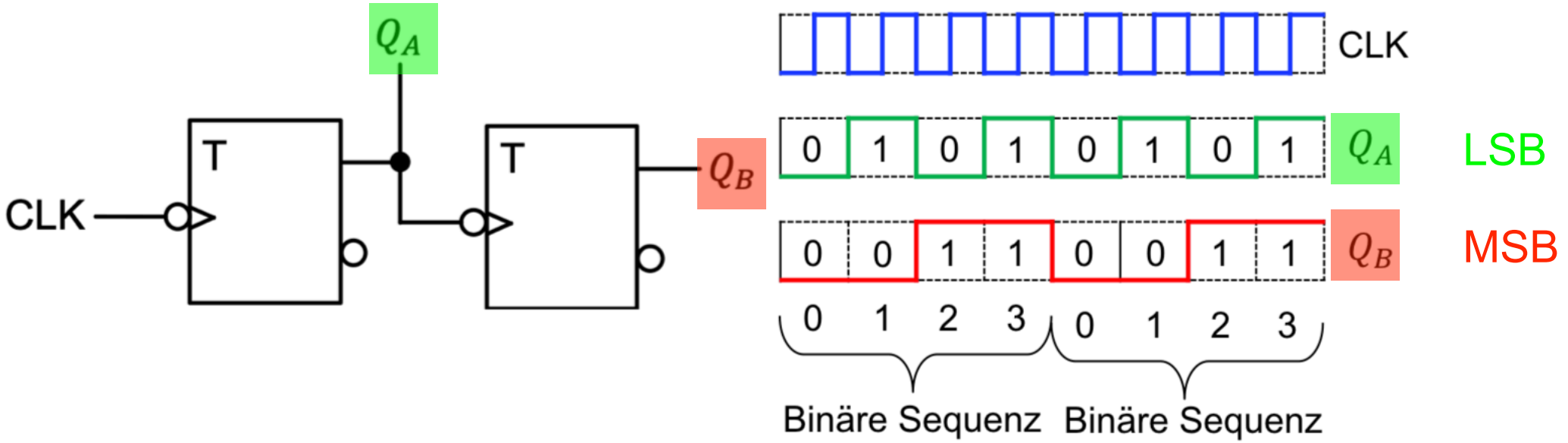
- T-Flipflops oder J-K-Flipflops mit J = K = 1
- n Flipflops in Serie führen zu einer Reduktion um 2^n





Dualzähler

- T-Flipflops oder JK-Flipflops mit J = K = 1
- Bei n Flipflops Zahlen bis 2^n - 1



Automaten

Automaten

Hier nur synchrone Automaten betrachtet (alle FF haben das gleiche Taktsignal)

Finite State Machine (= Endliche Automaten)

- mögliche Eingabezeichen
- mögliche Ausgabezeichen
- intern gespeicherte Zustände

} sind endlich

Beschreibung von Automaten

- X = Eingabealphabet
- Y = Ausgabealphabet
- Z = Zustandsmenge
- Z_0 = Anfangszustand
- F_c1 = Übergangsfunktionen
- F_c2 = Ausgangsfunktionen

Mealy - Automat

- Ausgang von Eingang und internem Zustand abhängig
- $Y_n = f_{C2}(X_n, Z_n)$

  $Z_{n+1} = f_{C1}(X_n, Z_n)$

Moore - Automat

- Sonderfall vom Mealy-Automat
- Ausgang hängt nur vom internen Zustand ab
- $Y_n = f_{C2}(Z_n)$

  $Z_{n+1} = f_{C1}(X_n, Z_n)$

- Medwedjew-Automat, wenn Ausgang = interner Zustand ($Y_n = Z_n$)

Zustandsfolgetabelle

$\nu_{max} = 2^{e+m}$

No	Eingang X_n	Momentaner Zustand Z_n	Folgezustand Z_{n+1}	Ausgang Y_n
1 ⋮ ν	$x_1, x_2, \cdots, x_e$	$z_{1n}, z_{2n}, \cdots, z_{mn}$	$z_{1n+1}, z_{2n+1}, \cdots, z_{mn+1}$	$y_1, y_2, \cdots, y_b$

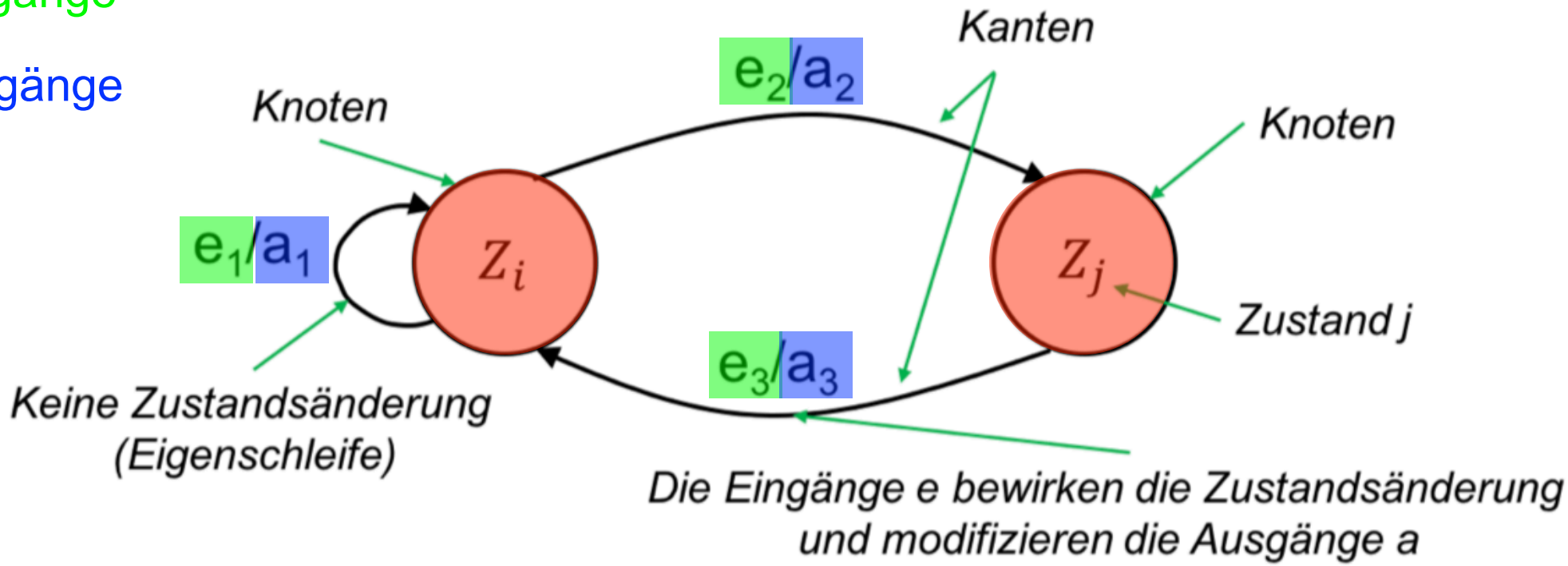
e+2m+b

Zustandsdiagramm – Mealy

Zustände

Eingänge

Ausgänge

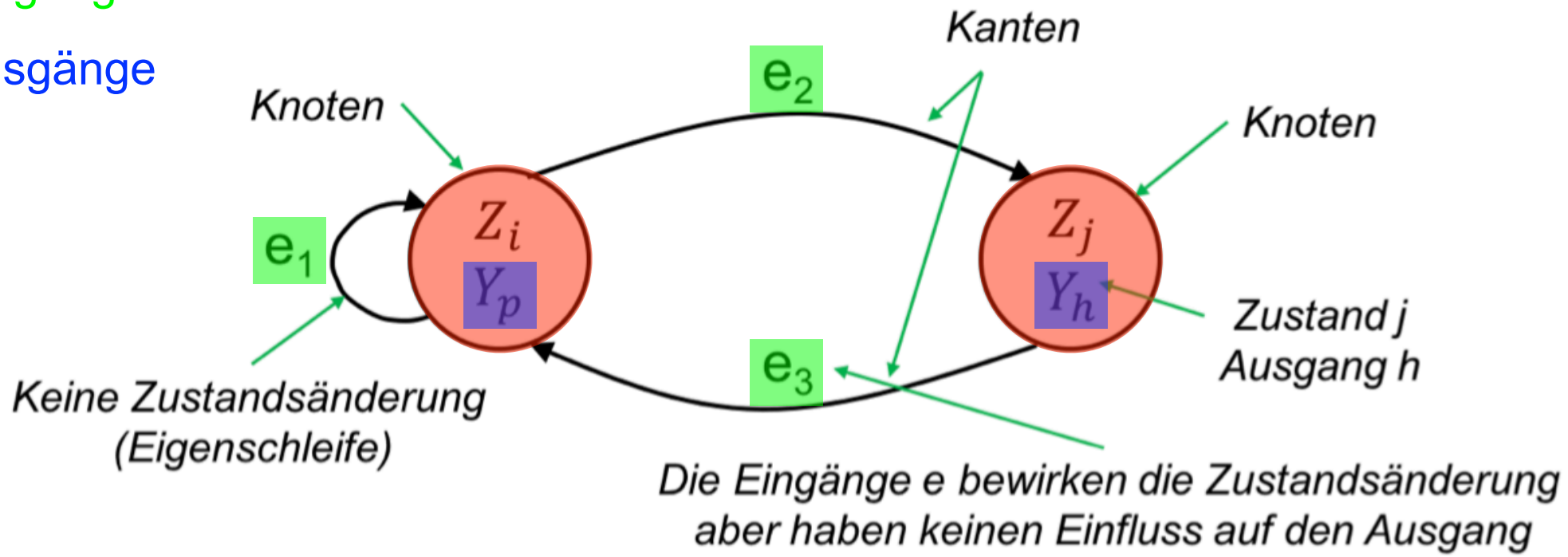


Zustandsdiagramm – Moore

Zustände

Eingänge

Ausgänge



Automatensynthese

- Zustandsmenge bestimmen
- Ein- und Ausgangvariablen bestimmen und Zustände kodieren
- Zustandsdiagramm zeichnen
- Zustandsfolgetabelle aufstellen
- Die Ausgangs- und Zustandsübertragungsfunktionen herausfinden
- Schaltplan zeichnen

Mealy - Moore - Umwandlung

- Mealy lässt sich immer in Moore umwandeln
- Einfach wenn die Zustände immer den gleichen Ausgang produzieren
- Sonst neue Zustände definieren

Dynamisches Verhalten

Mealy-Automat

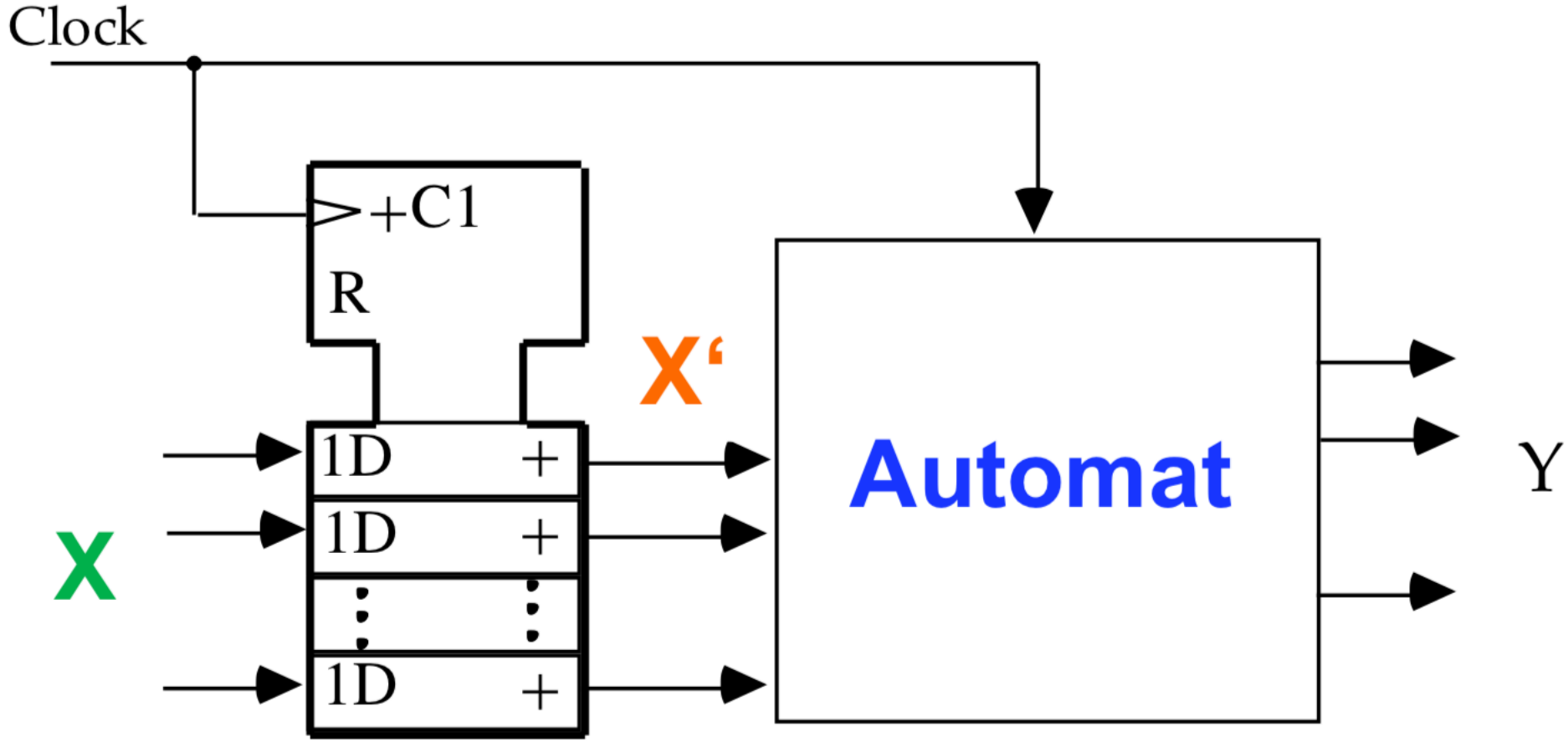
- Änderung des Eingangs wird sofort am Ausgang sichtbar

Moore-Automat

- Änderung des Eingangs wird erst bei der nächsten Taktflanke am Ausgang sichtbar

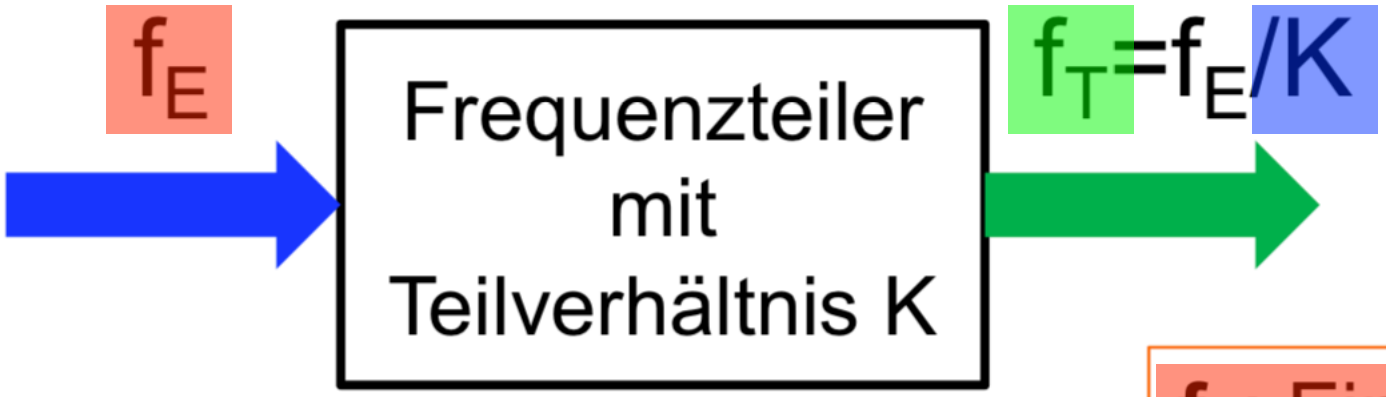
Synchronisierung der Eingänge

- Eingänge werden gespeichert und erst mit der nächsten Aktiven Taktflanke eingegeben



Frequenzteiler

- Zum reduzieren von Frequenzen



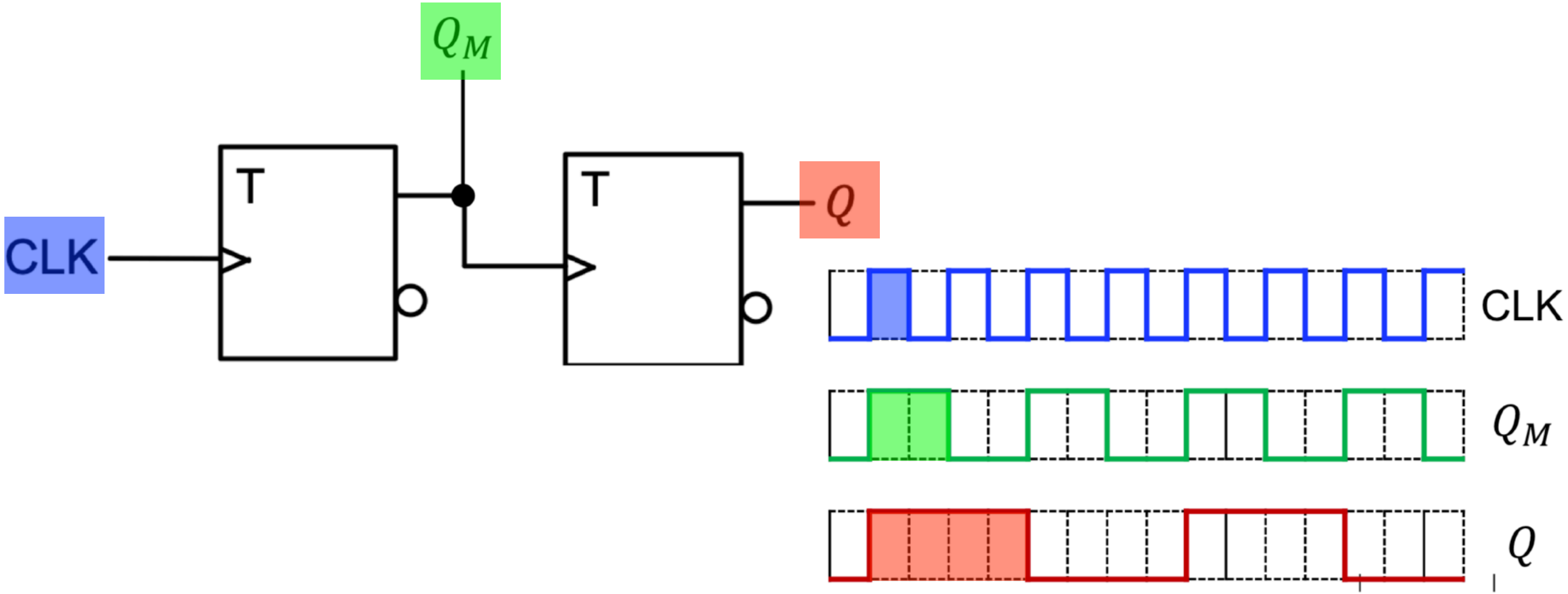
f_E: Eingangsfrequenz

f_T: Geteilte Ausgangsfrequenz

K: Teilverhältnis

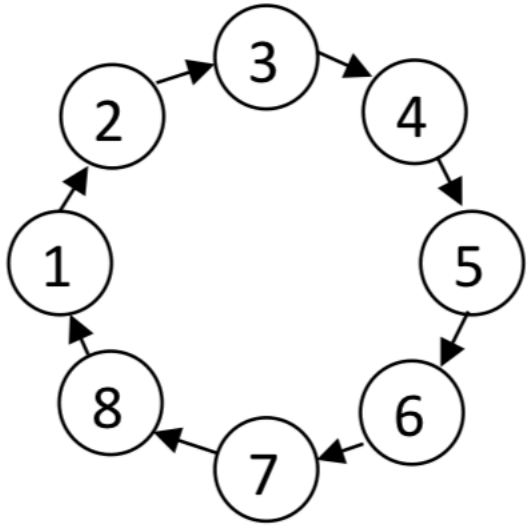
Flipflops als Frequenzteiler

- n Flipflops führen zu einer Reduktion von 2^n



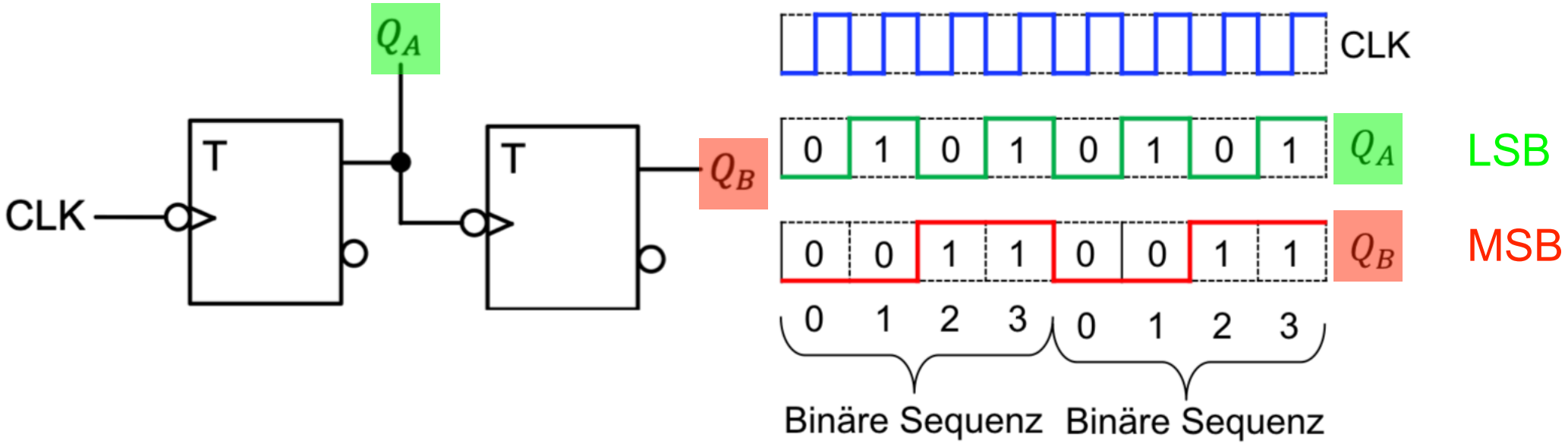
Zähler

- Zählen bis zu einer bestimmten Zahl und fangen dann wieder von vorne an
- Vorwärts zählen = +1 in jedem Schritt
- Rückwärts zählen = -1 in jedem Schritt



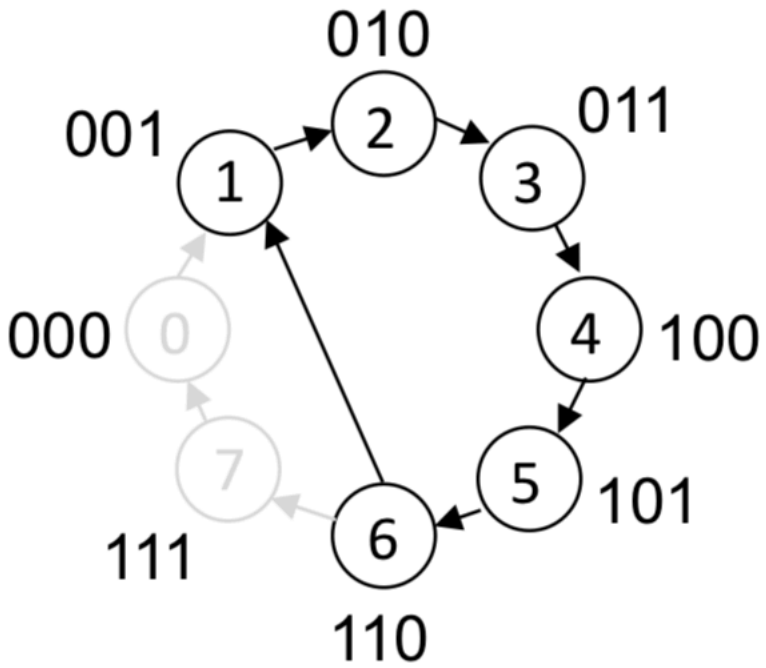
Dualzähler = Asynchronzähler

- Kaskadierung von T-Flipflops
- Mit n Flipflops kann man bis 2^n-1 zählen
- Einzelverzögerungen der Flipflops kumulieren sich ⟶ Asynchron
- Maximale Taktfrequenz: $f_{max} = \frac{1}{\sum_{i=1}^{n} t_{pd,i}}$



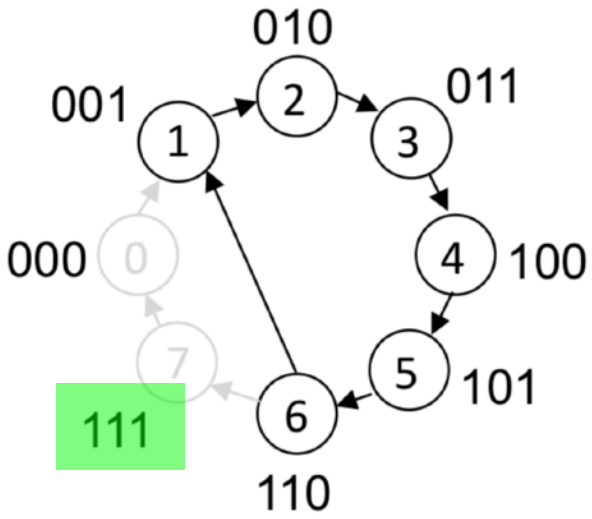
Modulo-n Zähler

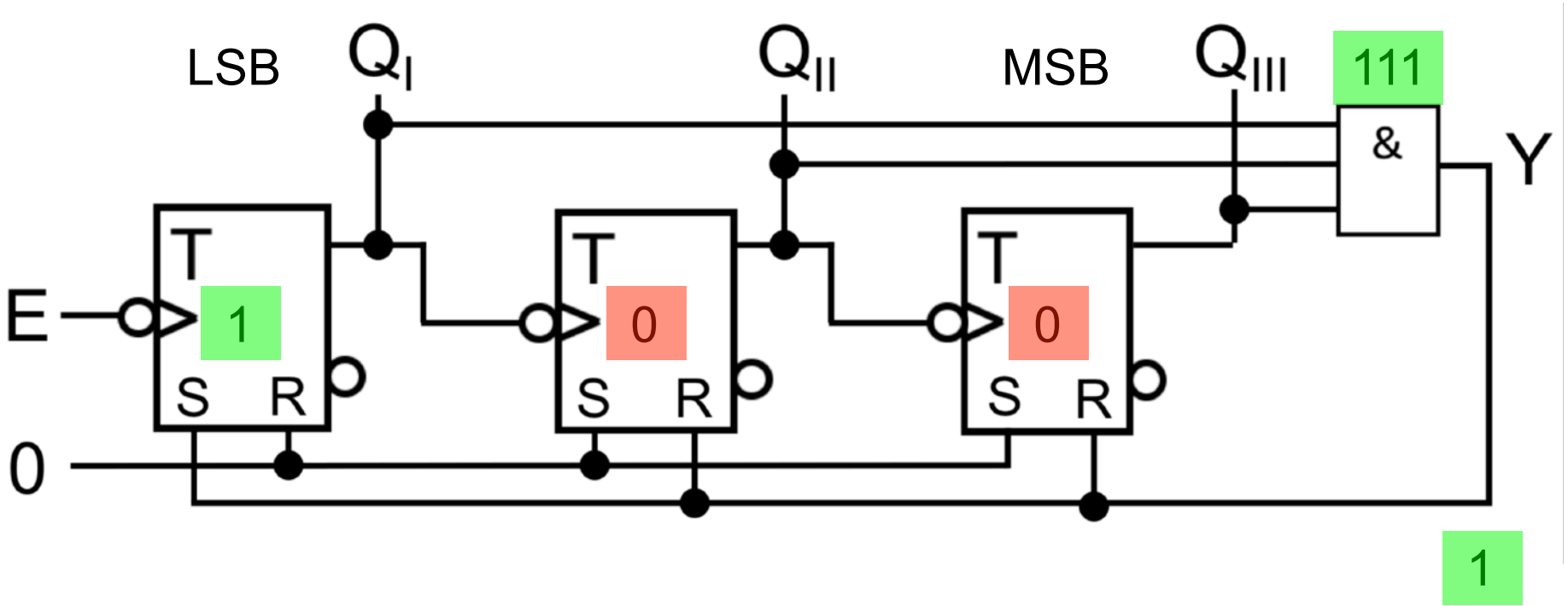
- Zählt bis n und springt dann zurück auf einen vorgegebenen Startzustand
- Umsetzbar als Asynchronzähler und Synchronzähler



Modulo-n Asynchronzähler

- Bsp.: Zähler von 1 bis 6
- Umsetzung mit Dualzähler





Modulo-n Synchronzähler

- Alle Eingänge liegen auf dem gleichen CLK (schalten gleichzeitig)
- Sind Medwedjew-Automaten
- Entwerfen wie Automaten

Zustandsgraph ⟶ Folgezustandstabelle ⟶ Karnaugh Diagramme ⟶ Schaltplan

Karnaugh Diagramm & Flipflops

D - Flipflops

- Karnaughdiagramm normal

JK - Flipflops

- Karnaughdiagramm mit Felder für Q und !Q

Q_{1n+1}

Q_1 \ Q_3Q_2	00	01	11	10
0				
1				

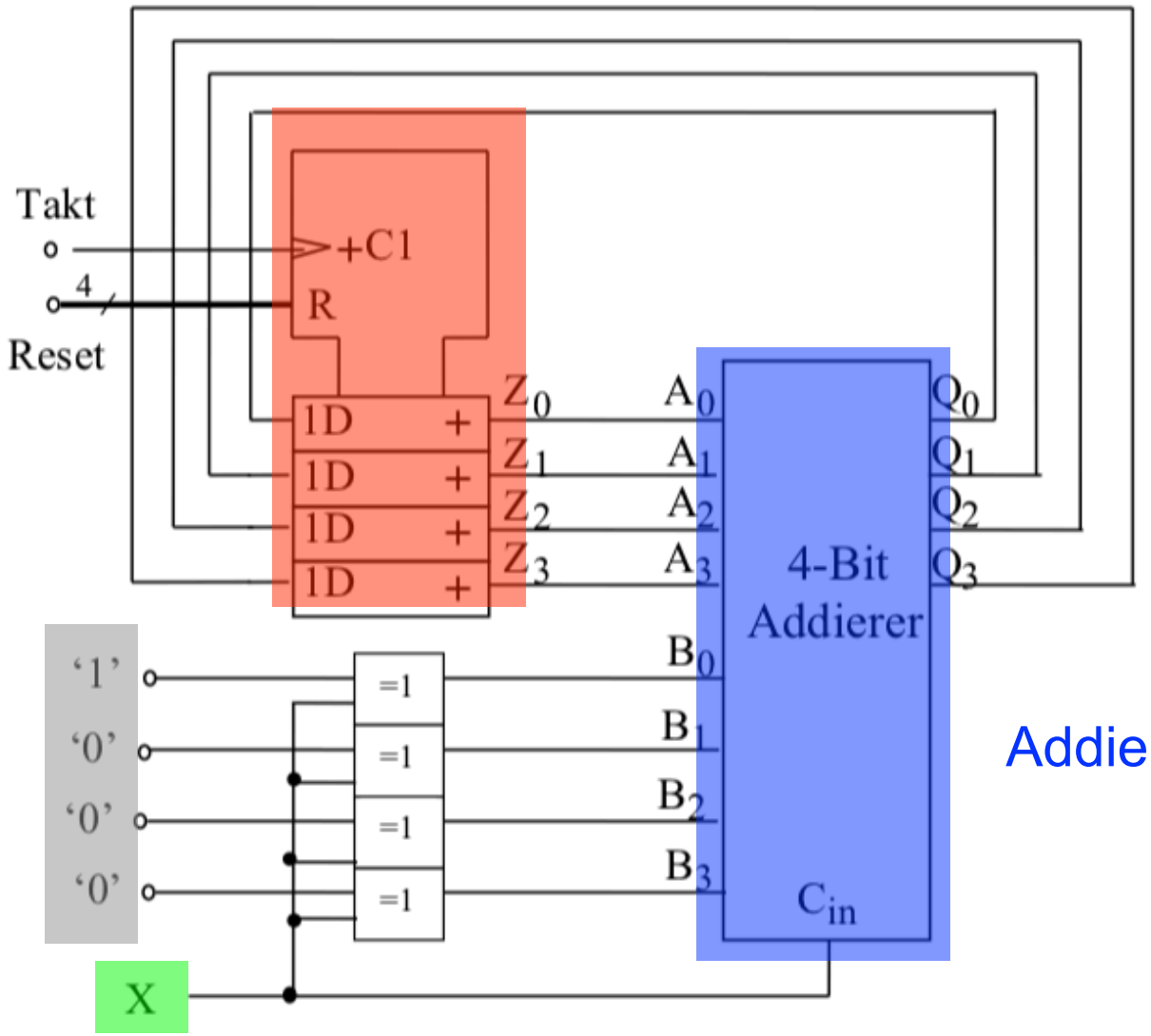
Q_1 \ Q_3Q_2	00	01	11	10
0				
1				

Q_{2n+1}

Reversible Zähler

- Zum vorwärts und rückwärts Zählen
- X = 0: Addition
- X = 1: Subtraktion

4 D-Flipflops mit gleichem CLK

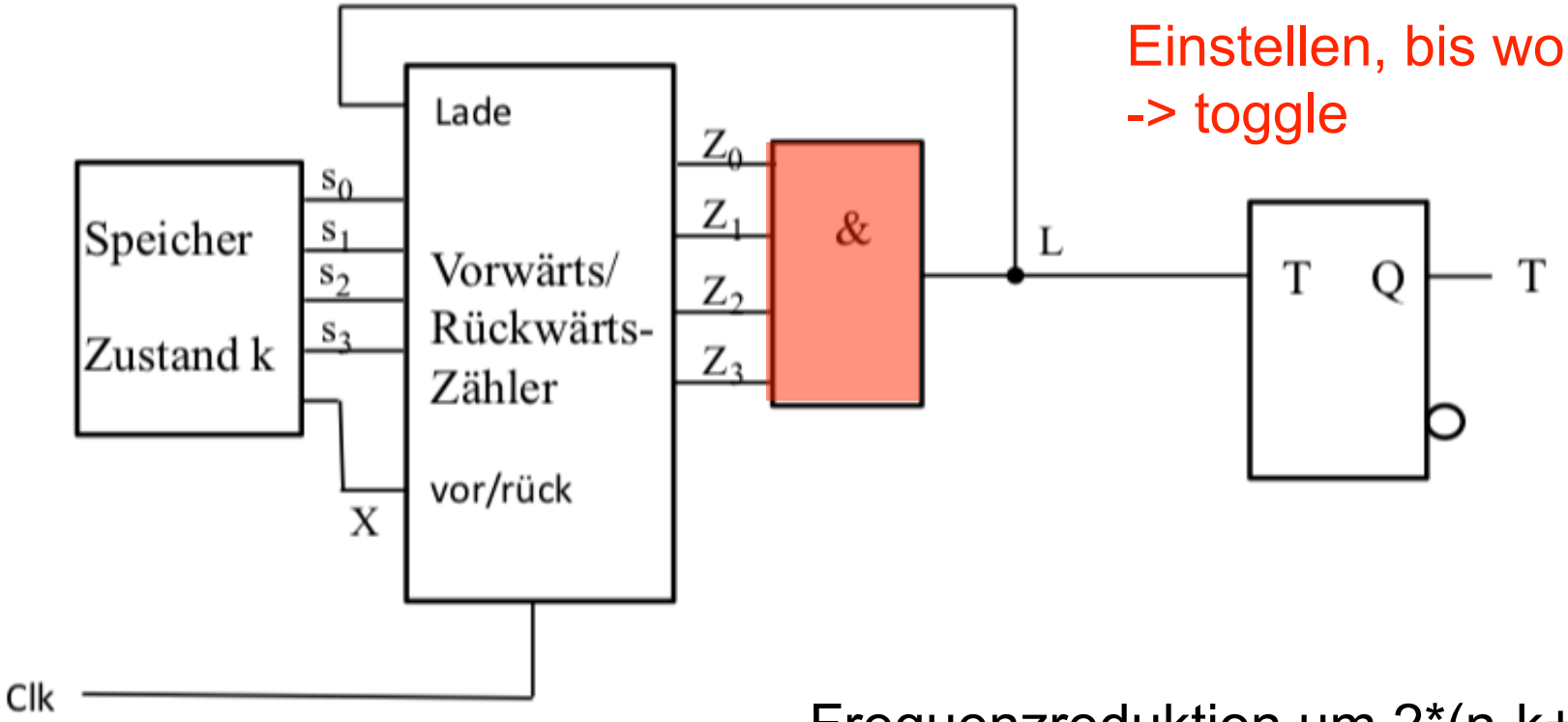


1 wird addiert (subtrahiert)

Addierer

Frequenzteiler mit Zähler

- Um Frequenz beliebig einzustellen



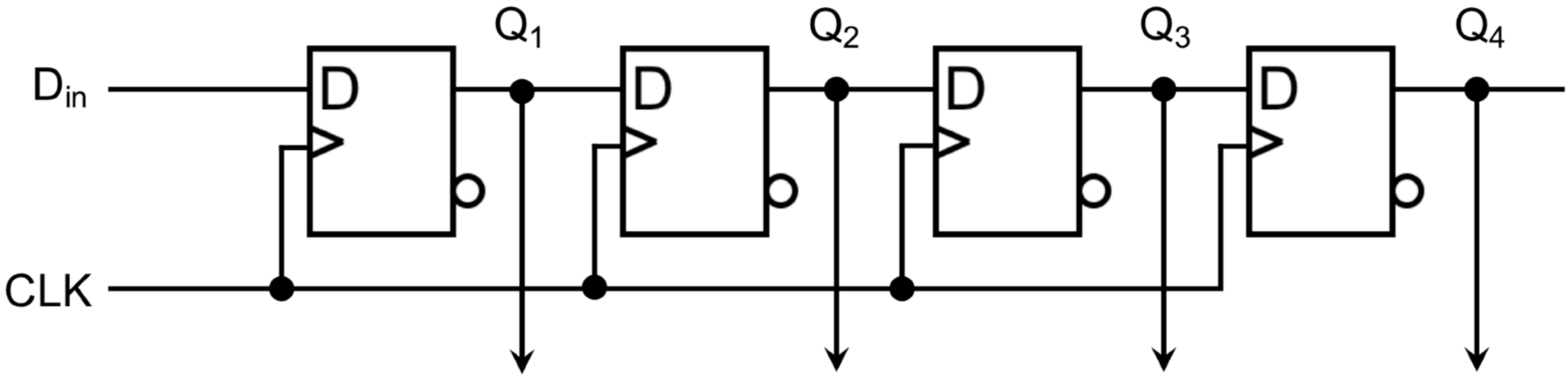
Einstellen, bis wohin gezählt wird
-> toggle

Frequenzreduktion um 2*(n-k+1)

Speicher

Schieberegister

- n = Anzahl Flipflops
  - n Bits können gespeichert werden
  - n Taktflanken um Bits einzulesen
- Nachteil: Teuer und Platzintensiv



Speichermedien

- Magnetband
- Plattenlaufwerk
- DRAM
- Flash

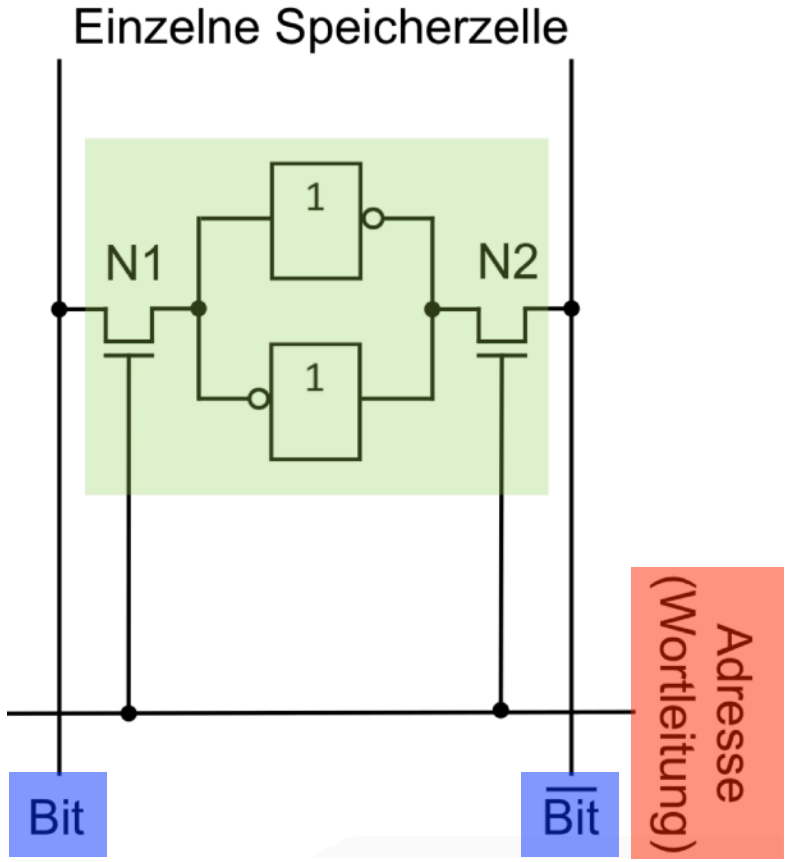
Speicherfunktionen

- ROM = Read Only Memory
- RAM = Random Access Memory

SRAM

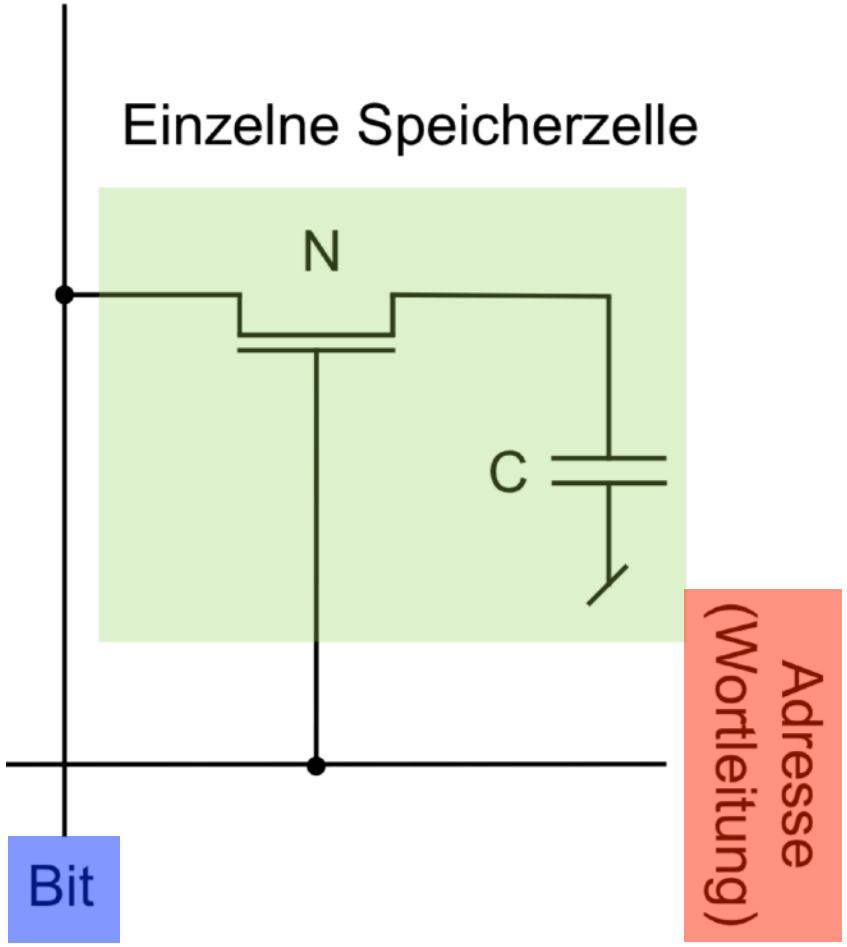
- static random access memory
- Speicherzelle wird angewählt
- Speicherinhalt wird gesetzt/gelesen

- lesen: Bit = 1, !Bit = 1
- 1 schreiben: Bit = 1, !Bit = 0
- 0 schreiben: Bit = 0, !Bit = 1



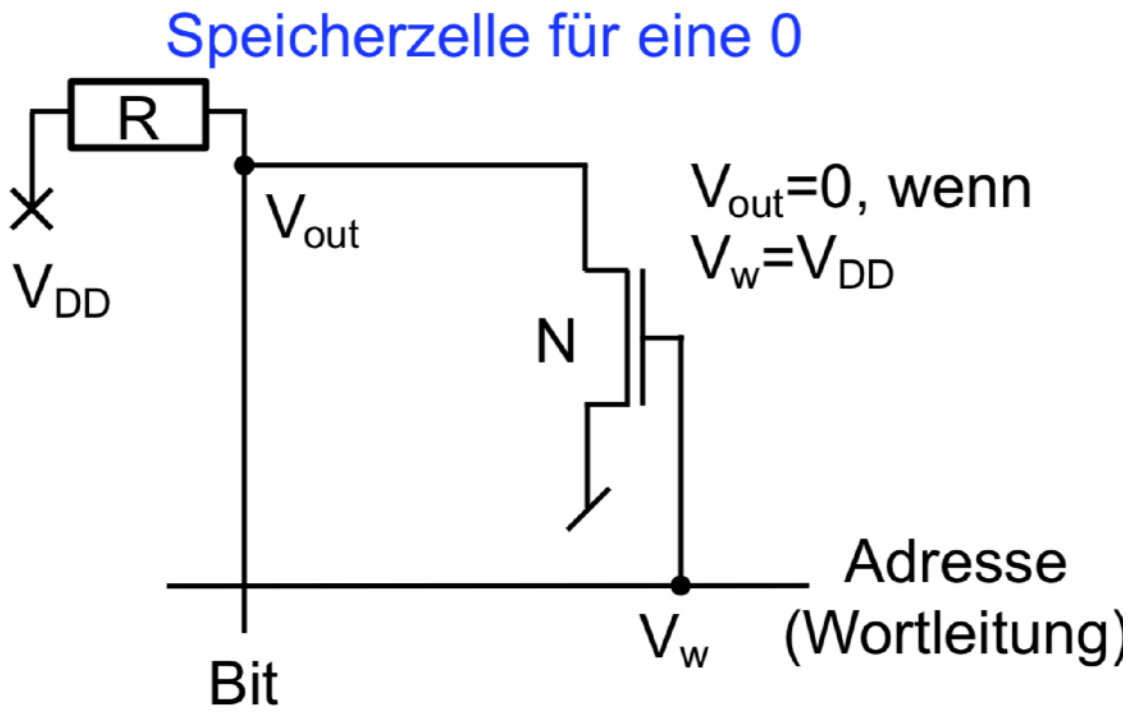
DRAM

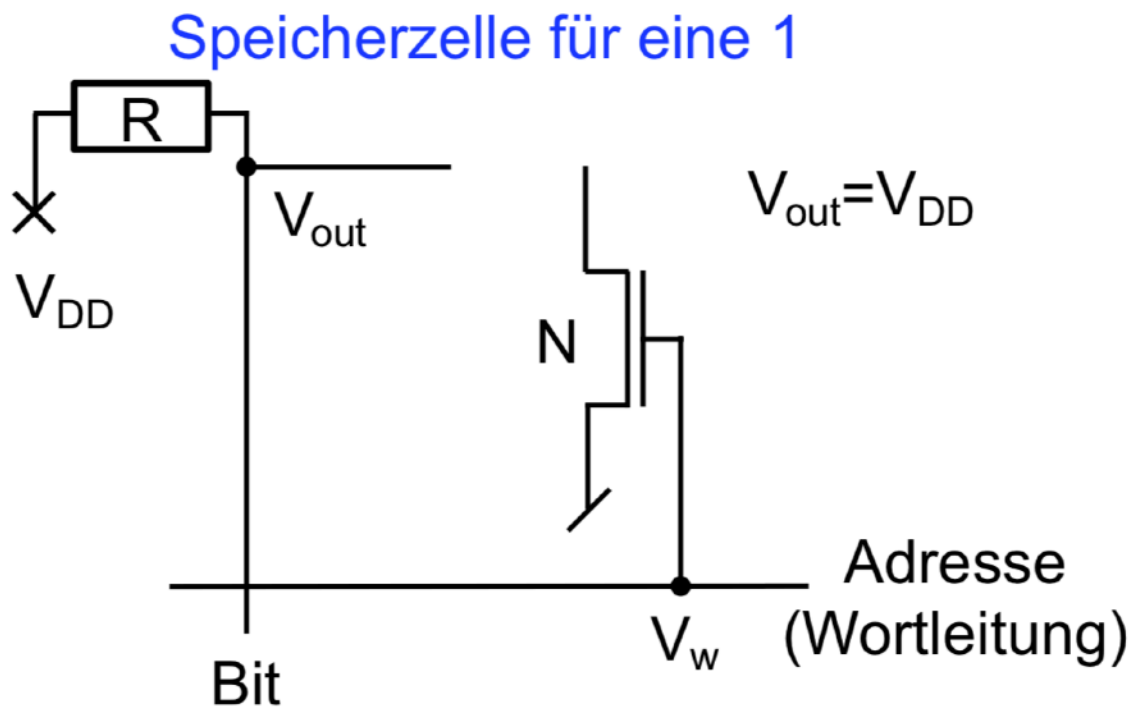
- dynamic random access memory
- Speicherzelle wird angewählt
- Speicherinhalt wird gesetzt/gelesen



ROM

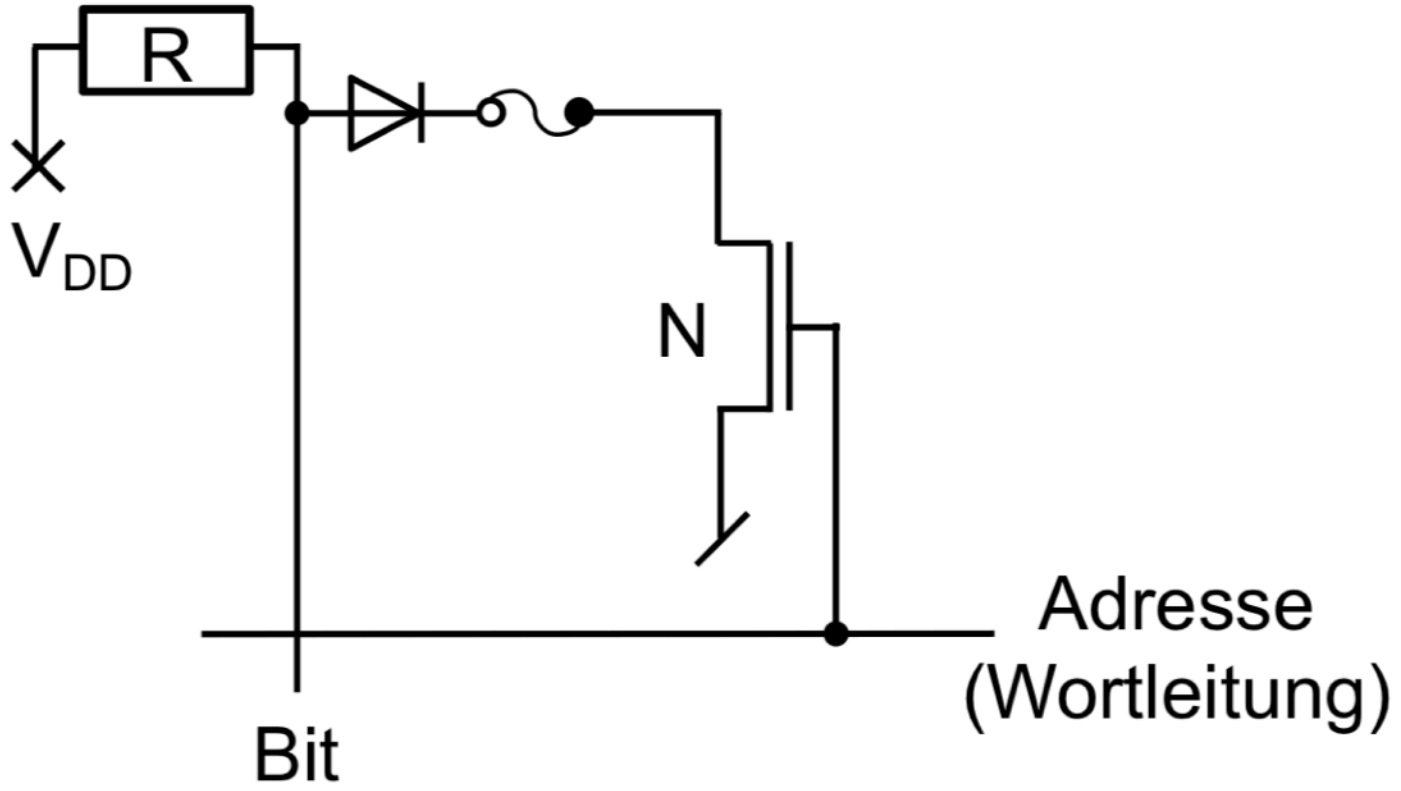
- Read only memory
- werden bei Produktion programmiert





PROM

- Programmierbare ROM
- Als 1 oder 0 programmierbar



Basisprüfung

Tipps

Tipps

- Macht euch klar, was eure Stärken sind.
- Ihr müsst nicht alles lösen.

Fragen?

Viel Erfolg!